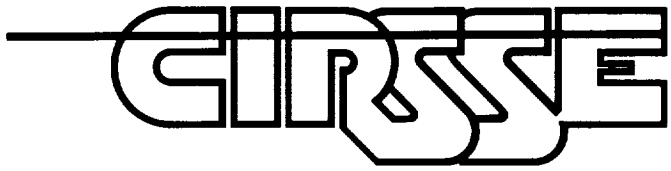
CIRSSE-R-78

NAGW-1333

Grant
NAGW-1333
153764
p 24



Center for Intelligent Robotic Systems for Space Exploration

Rensselaer Polytechnic Institute
Troy, New York 12180-3590

A CONNECTIONIST/SYMBOLIC MODEL FOR PLANNING ROBOTIC TASKS

by

Michael Craig Moed

Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering
Troy, New York 12180-3590

December, 1990

CIRSSE REPORT #78

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I wish to express my sincere gratitude to Robert Kelley, for the advice, encouragement and support that he provided during the years I have spent at Rensselaer.

Special thanks go to Chuck Stewart, who sparked my interest in Genetic Algorithms and worked with me to develop many of the concepts addressed in this thesis. Thanks also goes to George Saridis who greatly aided in my development as a PhD candidate and a colleague.

I would like to thank all my committee members for providing the guidance and direction that allowed me to develop this work.

I am indebted to Robert O'Bara and Glenn Tarbox, primarily for their valued friendship, but also for reading drafts of this thesis. Thanks also goes to Philippe Jacob for his friendship during my Pennsylvania and Rensselaer tenure.

I would also like to acknowledge all the members of the old RAL and of CIRSSE who made our labs an enjoyable place to work.

It is difficult to describe in these few lines the love and encouragement provided by my parents, Annette and Richard, and my brother, Edward, throughout the years. Without their support, I would not be where I am today.

Finally, I thank my fiancee', Deborah, for her love, support and automobile driving abilities. The last two years we've spent together make me look forward to our next 50.

ABSTRACT

This work develops an evaluation system, called the Associative Rule Memory (ARM), designed to operate with an interactive or automatic planner in a robot-based world model. The ARM ranks alternative robotic actions based on the probability that the action works as expected in achieving a desired effect. The system is experience-based, and can predict the probability of achieving a desired effect for robotic actions that have not been explicitly tested in the past. The ARM is constructed to quickly and efficiently find high probability of effect robotic actions for a given desired effect. The design of the ARM is based on a neural network called the Boltzmann Machine, which is adapted for this work. An algorithm is presented for training the ARM on tested robotic actions, and it is shown to globally converge to an accurate representation of the training set. Also, the network is able to develop hidden nodes that represent higher order relationships through the training procedure. The Genetic Algorithm (GA) is used for associative recall on the ARM, and is shown to be applicable to searching a Boltzmann Machine. An immigration operator is added to the GA, and the modified GA is shown to be more efficient on a test suite of functions. A proof is constructed that guarantees that the GA with immigration will converge in probability to the optimum of a given function. The use of the ARM as the Organization level of the Intelligent Machine is demonstrated. The functions of the ARM are tested in the world of the NASA Flight Telerobotic Servicer and results detailing accuracy and efficiency are presented.

CHAPTER 1
INTRODUCTION

1.1 Introduction

Automatic, computer-based systems have been developed to mimic the planning capabilities of people. These systems plan in an abstract model of the world that does not capture all the features and details of the real world. The planners concentrate on the significant aspects of objects in the world to reduce the difficulty of plan formation. It is important, however, that the abstract model contain sufficient detail, so that an automatically generated plan can be executed reliably in the real world.

Some systems plan by using a divide and conquer strategy to recursively separate plan goals into subgoals, and then attempt to solve each subgoal of the plan. Other systems plan in successive levels, by developing high level plans first, and recursively expanding, detailing and ordering each item in the plan. Some systems assign a cost to each step, and build a plan from the sequence of steps minimizing the total plan cost.

In many world models, there exist several ways to accomplish a step in a plan. If the world is sufficiently complex, the number of possible alternatives can become quite large. When developing plans in such a world, the size of the search space for the plan becomes combinatorially huge.

When people plan, they are often faced with a large number of possible alternatives for each plan step, yet are usually able to develop a sequence of planning steps quickly and efficiently. People are able to rule out many alternative steps by using previous experiences to determine which candidate steps will work well. When specific experience is lacking in a particular domain, a person attempts many

alternative plans, trying to relate previous experience to the new situation, until a desired goal is achieved.

To limit the number of alternative steps in an automatic planning system, an experience-based evaluation function can be applied to rank each of the candidate steps according to some optimality criteria. Selecting the optimal step, or set of near-optimal steps for possible plans would greatly reduce the planning search space.

It is possible that past experience may not cover a particular planning step. In this case, it is necessary to predict the optimality of candidate steps by using related experience of similar steps.

Since the number of possible steps can be quite large, however, ranking each alternative can be rather time consuming. Instead, a search technique may be employed to find an optimal step, or set of near-optimal steps quickly and efficiently.

This thesis develops an evaluation system designed to operate with an interactive or automatic planner in a robot-based world model. The planning step in this model is called a robotic action. The change that it should produce in the world model is called a desired effect. The evaluation system ranks a candidate robotic action based on the probability that it works as expected in achieving its desired effect. The system is experience-based, and can predict the probability of achieving a desired effect for robotic actions that have not been explicitly tested in the past. A search technique is also present in the system that quickly and efficiently finds optimal robotic actions for a given desired effect.

1.2 Problem Statement

For a sufficiently complex world, an interactive or automatic planning system may contain a large number of robotic actions that can achieve the same effect on the world model. Since a large number of robotic actions leads to a large planning search space, a method must be developed to reduce the number of candidates, for

a given desired effect. This method must:

1. Receive a desired effect as input from a user or automatic planning system.
2. Evaluate and rank the robotic action candidates according to some optimality criteria.
3. Quickly and efficiently search the set of candidates for an optimal robotic action, or set of near-optimal robotic actions that achieve the desired effect.
4. Output the optimal robotic action. or a set of near-optimal robotic actions, along with their evaluation function values.

For this study, we designate the optimality criteria to be the probability that a robotic action achieves a desired effect. This will be called the probability of effect value. Using this criteria with a sufficiently complex world model, it is apparent that it is difficult to test all possible sets of robotic actions and effects to determine their probability of effect values. Instead the method must:

1. Be provided with a set of tested robotic actions and effects along with corresponding probability of effect values. This is called a training set.
2. Use this training set to predict probability of effect values for untested robotic action/effect pairings.

The method must also show applicability to planning systems, including the Organization level of the Intelligent Machine [1]. Finally, this method must demonstrate these capabilities through a case study using a target robotic environment.

1.3 Method of Approach

The following approach is used to develop a system that solves the problem stated above.

1. A set of system requirements and constraints are outlined.
2. A grammar is defined to allow the system to interface with a user or automatic planner.
3. Several models are considered to form the evaluation function, which must be trainable and provide predictive probability of effect values.
4. An evaluation function model is chosen based on its capabilities and described in detail.
5. The architecture of the chosen model is specialized to fit the constraints of the above problem.
6. A method for training the model to a desired degree of accuracy is developed.
7. The model is shown to demonstrate the ability to reliably predict probability of effect values for untested robotic action/effect pairings.
8. Different techniques for searching for robotic actions given a desired effect are examined given the constraints of the developed model.
9. Methods for specializing and accelerating the chosen technique are developed.
10. The search technique is modified so that it can provide sets of high probability of effect robotic actions as output when given a desired effect as input.
11. A case study involving a complex target robotic environment is developed and the capability of the system to perform as desired is demonstrated.

1.4 Organization of the Thesis

The thesis is organized as follows.

- Chapter 2. This chapter provides an in-depth introduction to the problem and also presents a literature review of recent research in related areas. The target world model for this thesis is described. Different evaluation functions are discussed and necessary capabilities for an evaluation function model, called the Associative Rule Memory, are defined. Input/Output and structural requirements are defined for the model. Candidate neural network models are reviewed to form the model of the evaluation function. A Boltzmann Machine is chosen based on its capabilities. Different optimization techniques are also reviewed.

- Chapter 3. This chapter develops the Associative Rule Memory model using the formulation of a Boltzmann Machine. The theory behind the Boltzmann Machine is reviewed and critiqued. A specialized architecture based on the Boltzmann Machine is developed to fit the requirements of the Associative Rule Memory. A technique for training the model is developed and shown to converge to the correct representation. Examples of training are provided and the ability of the Associative Rule Memory to predict probability of effect values for untested robotic action/effect pairings is shown.

- Chapter 4. This chapter examines search techniques for the Associative Rule Memory. The requirements for a search technique are outlined based on the constraints of the Associative Rule Memory. Simulated annealing and the Genetic Algorithm are compared for search efficiency. A method to reduce the search time of a Genetic Algorithm, called immigration, is described and experiments are discussed. A proof is developed to show that a Genetic Algorithm enhanced with the immigration operator will converge in probability to the global optimum of a cost function. Representation issues are examined. Also, modifications to the Genetic Algorithm to allow it to find sets of

solutions are outlined.

- Chapter 5: This chapter reformulates the Organization level of the Intelligent Machine as a Boltzmann Machine and demonstrates that the Associative Rule Memory can be used to form this level. An introduction to Intelligent Machines is provided, along with a mathematical description of the Organization level.
- Chapter 6: This chapter provides a case study using the Associative Rule Memory in the world of the NASA Flight Telerobotic Servicer.
- Chapter 7: This chapter discusses the overall system and presents the conclusions of this thesis.

1.5 Contributions

The main contributions of the thesis are as follows.

1. The development of a model that will produce a set of optimal robotic actions as output, given a desired effect as input.
2. The design of a neural network model that is able to represent a symbolic grammar comprised of a robotic action and effect.
3. The ability of this model to maintain instantiations of the grammar with a real valued number representing the probability that the robotic action achieves the desired effect.
4. A training procedure that guarantees that the network will develop accurate probability of effect representations for all robotic action/effect pairings in the training set.
5. A training procedure that develops weighted connections in a neural network that represent the extent to which a robotic action symbol affects an effect symbol.

6. A technique for adding higher order nodes to a neural network model when necessary, and pruning them when they are unnecessary.

7. A demonstration that the training procedure builds neural network connections that can be used for predicting probability of effect values for untested robotic action/effect pairings.

8. The development of the immigration operator for Genetic Algorithms and the demonstration that immigration improves the performance of a Genetic Algorithm on functions that possess difficult local optima.

9. The proof that a Genetic Algorithm combined with the immigration operator will converge in probability to the global optimum of a cost function.

CHAPTER 2
PROBLEM INTRODUCTION AND LITERATURE REVIEW

Planning systems, whether automatic or interactive, attempt to develop a set of steps that change the world model, or environment, from an initial state to a goal state. Each step in the plan dictates an action, a set of agents that perform the action, and a set of agents on which the action should be performed. In environments that possess a large number of agents as well as a large number of possible actions, the number of possible alternatives for any step in a plan becomes combinatorially large. To reduce the planning search space that these alternatives produce, an evaluation function can be applied to the set of alternatives, and a rank can be assigned to each choice based on the likelihood that the choice will lead to a successful plan. By selecting the best alternatives as possible planning steps, the combinatorial explosion of plan choices is eliminated.

This chapter begins the development of a particular planning evaluation technique, called the *Associative Rule Memory* (ARM), which is designed to rank and select steps in a robotic planning system. The purpose of this chapter is twofold. First, it serves to introduce the problem that this thesis addresses. Second, it presents a literature review that describes recent research in related areas.

Section 2.1 of this chapter presents the target world model for our system. Section 2.2 presents several automatic planning systems, and determines the necessity of an evaluation function for large numbers of alternative steps. Section 2.3 discusses different evaluation functions and dictates the capabilities that must be possessed by an evaluation function. Section 2.4 examines the input/output and structural requirements placed on an evaluation function to allow it to work within a planning system. Section 2.5 reviews different neural network models that can be used for the ARM. Optimization techniques for the ARM are presented and reviewed in section

2.6. Conclusions are presented in section 2.7.

2.1 Target World

The target world for this study is composed of

1. A set of *actors* which includes
   (a) Manipulators
   (b) Positioners
   (c) Transporters
2. A set of *objects* which includes, but is not limited to
   (a) Tools
   (b) Platforms
   (c) Pallets and Carriers
   (d) Bays and Loading sites

It is possible for an actor to be used as an object. Together, actors and objects are referred to as *agents*.

Each agent possesses a set of *states*. that describes features of the agent in the world model. An actor can perform an *action* on an object, perhaps by using another object, and the result is a change in the state of the first object. When this occurs, the combination of actor, action and objects is said to perform a *robotic action*. The change of state of the first object is said to be the *effect*. In the target world, it is possible for many different robotic actions to achieve the same effect. This allows redundancy in the world, which adds extra freedom to that way that tasks can be performed.

The agents and actions of the world have been abstracted for planning and modeling purposes, and are represented by symbols. The actor symbols are referred

to as *ACTORs*, object symbols as *OBJs* and action symbol as *ACTIONs*. A list of symbols that are used in this study are presented in Appendix A. The agents and actions used in this study are based on the world of the NASA Fight Telerobotic Servicer Task Analysis Methodogy [2], as further described in Chapter 6.

2.2 Planning Models

One of the earliest planning systems was STRIPS [3]. The world model for STRIPS is a set of first-order predicate calculus well-formed formulas (wffs) that represent the state of agents in the world. STRIPS also consists of a set of operators that, when applied, transform the world model into a new world model. Each operator is composed of three parts: conditions, action, and effects. The conditions dictate when the operator can be used and are comprised of wffs. The action is simply a string representing the action in the real world. The effects dictate the changes in the world model if the action is performed, and are a set of wffs that determine which items from the world model should be added or deleted.

STRIPS combines Means-End Analysis [4] with logic resolution techniques to develop a plan of actions to change the initial world model to a goal world model. The basic procedure is as follows.

1. Find the wff agent state differences between the current world model and a (sub)goal world model.

2. Construct a set of operators whose effects will eliminate some of these differences.

3. Select one operator from the set of candidates, instantiate it with agent symbols available in the world model, and form new subgoals from the conditions of this operator.

4. Eliminate those subgoals that can be resolved from the current model.

5. If the current model is not the same as the goal world model, go to 1.

Step 3, the selection and instantiation of an operator for application, is relevant to our discussion. In STRIPS, a candidate operator is selected if the clauses on its effect list can resolve away difference clauses between the current and (sub)goal world models. The operator is then instantiated with agent symbols that allow this resolution to occur.

For a reasonably complex environment, such as our target world model, there may exist many operator instantiations whose add list can resolve away the same difference clauses. This is due to the fact that in our target world, many different robotic actions may lead to the same set of effects. This leads to a very large number of candidate operators, all of which appear appropriate.

If each candidate operator is selected for application, the search tree would grow geometrically. To prevent this, only a small subset of candidate operators should be selected for application. To reduce the number of possible operators, each must be evaluated and ranked according to some defined criteria. The highest ranked operator or operators could then be selected for application. STRIPS does not provide this functionality. Therefore. constructing a plan using STRIPS in our target world may lead to a combinatorial explosion in the planning search space.

Another problem of STRIPS is that it can not be guided to solve the main part of a plan first, and reserve the details for later planning. This leads to excessive search times when developing plans because each detail of a plan is treated with equal importance. To overcome this limitation, ABSTRIPS [5], was created to develop plans in successive levels of detail.

ABSTRIPS is based on the STRIPS model. To plan at successive levels of detail, ABSTRIPS ranks each precondition of an operator as a function of its importance in a plan. Preconditions are assigned high values if they are necessary for the main part of a plan. Preconditions are assigned low values if they are considered

details. For each level of detail, or abstraction level, ABSTRIPS uses Means-End Analysis to create plans by using only those preconditions of operators whose value is greater than that level. By creating high level plans and successively refining them at lower levels, an overall plan is developed in significantly less time than STRIPS would require.

Sarcedoti states

> "A good heuristic evaluation function will enable a problem solver to reject most of the possible paths in a situation space".

The heuristic evaluation function he chose was to rank the preconditions to eliminate search tree branching at each abstraction level; however, another evaluation function is also required. As one can see, ABSTRIPS suffers from the same malady as STRIPS if used to plan in our target world. The elimination of preconditions for abstract planning does nothing to reduce the number of possible robotic action alternatives that our target world provides at each planning step and in each abstraction space.

NOAH [6] moved away from the predicate calculus, Means-End analysis planning style and introduced "procedural nets" to produce plans with nonlinear constraints such as operations that have to be time ordered for successful execution. Each procedural net contains an action at some level of detail, along with "fork" and "join" nodes for combining actions that can be executed in parallel. As the net is expanded level by level, a set of knowledge-based critics examine the actions at the current level and dictate those that must be ordered, and those that can execute in parallel. When the bottom level of the net is reached, the action nodes correspond to actions that must be performed by agents in the given world.

When creating robotic actions, NOAH will avoid binding an *ACTOR* or *OBJ* variable to an agent symbol until it is absolutely necessary. This least-commitment

approach adds a degree of flexibility to planning, since it allows uninstantiated variables to be set by other system constraints, such as resource availability. Sarcedoti suggests employing a "Use Existing Objects" critic, which conserves resources by binding variables to agent symbols that have already been used in planning.

For a reasonably complex environment, such as our target world, many of the variables will be left uninstantiated at the final level of the plan, since many robotic actions can achieve the desired effects needed by the plan. The "Use Existing Objects" critic will not be able to bind all the uninstantiated variables, so another critic, in the form of an evaluation function, is needed to select the *ACTORs*, *ACTIONs*, and *OBJs* that should be used.

Other planning systems followed that use some of the ideas presented in NOAH. MOLGEN [7, 8] combines hierarchical planning and a least-commitment approach with variable constraint passing. As planning steps are developed, MOLGEN develops a constraint list that limits the agents that can be used to instantiate variables in planning steps. The constraints aid the planning system by reducing the size of the search space for each planning step. NONLIN [9] adds backtracking to NOAH to reduce the possiblity that a suitable plan is not found.

The SIPE [10] planner is also based on the NOAH system, but is one of the first planners to integrate feedback into planning. If a failure occurs during the execution of a generated plan, a SIPE module examines the state of the world after the failure and replans the task from the current state. SIPE also adds MOLGEN-like constraints and backtracking to the NOAH system.

Recently, Rokey [11] developed a two-level planner for the JPL telerobot testbed. The planner, TIPS, draws strongly from NOAH and SIPE, and is designed to operate in a multiagent, redundant world. Stage 1 of the planner employs procedural nets to develop a set of largely uninstantiated actions that achieve a given goal. Stage 2 of the planning system binds *ACTOR* and *OBJ* symbols to

uninstantiated variables using opportunistic scheduling heuristics. In this stage, the agents are viewed as resources, and available resources can be assigned to a given action, much in the same way a computer operating system allocates resources to running processes. Resources that are not available form the set of constraints for each planning step. Stage 2 allows the planner to recover from unexpected failures by reallocating resources on-line, if necessary.

In effect, the scheduling heuristics form an evaluation function that ranks possible robotic actions, and assigns higher values to those that represent under utilized agents. It is quite possible for many possible robotic actions to achieve high rankings after evaluation, if their agents are available for use. This is particularly true in a highly redundant setting, such as our target world. The addition of a second evaluation function can be used to reduce the number of candidate robotic actions that achieve the desired effect that has been provided by Stage 1 of the plan.

Saridis et al. [1, 12, 13, 14, 15, 16] have developed the concept of an "Intelligent Machine" that combines elements from the disciplines of Artificial Intelligence, Operations Research and Systems Theory. The Organization level of the Intelligent Machine, as discussed in [17, 18], is responsible for high level planning activities. The function of the Organization level is to form activity strings from primitive events and order the activity strings to form plans. Under this theory, the primitive events represent abstracted actors, actions and objects in the world.

The theory of Intelligent Machines emphasizes a mathematical approach to planning by selecting activity strings that minimize a system cost function. This cost function is the Entropy of the activity, and represents the uncertainty of the activity. In other words, the Entropy function forms the evaluation function that eliminates candidate robotic actions that are highly uncertain.

Valavanis [17] proposed one algorithm for selecting primitive events for activities based on their Entropy values. Unfortunately, the suggested algorithm can be

computationally expensive, especially for large numbers of possible activities. This places a second requirement on possible evaluation functions: If a large number of candidate robotic actions exist for a given planning step, the evaluation function must *efficiently* determine a subset of candidates that are optimal with respect to the evaluation function.

2.3 Evaluation Functions for Robotic Planning Systems

At each step in a plan, a robotic action is used to change the state of the world. In a complex world, many robotic actions may lead to the same effect. For a given effect, an evaluation function can be applied to the set of applicable robotic actions to determine which candidates from the set are "optimal" in some sense. The non-optimal candidates can then be eliminated and only the optimal robotic action(s) will be used as a planning step.

Many different optimality criteria can be applied to robotic actions. Some of these are:

- Uncertainty in achieving the desired effect.
- Complexity in achieving the desired effect.
- Time to complete desired effect.

Each criteria can rank the set of robotic actions to determine an optimal one for the given desired effect. The evaluation function that we consider in this work is the uncertainty of a robotic action in achieving a desired effect.

Consider a world with many actors, objects, and different actions that can be performed by the actors on the objects. To achieve a desired effect, an actor, action and a set of objects must be selected to form the robotic action. From the structure of the robotic action, many agents and actions may be possible.

It is likely, for example, that certain actors work better than others at achieving a particular effect. It may also be the case that certain tools cannot be reliably used to achieve a particular effect, while other similar tools can. In some situations, an actor and object may not work well together in achieving an effect. All of these factors contribute to the uncertainty that a robotic action successfully achieves a desired effect.

It would be extremely useful if, given a desired effect, the evaluation function could examine the set of symbols contained in a candidate robotic action and produce a value representing the probability that the robotic action causes the desired effect. This probability will be referred to as the *probability of effect* (POE) for a robotic action and a desired effect. The POE value could be developed through experimentation in the environment and stored in a database. It would be even more useful if a desired effect could be provided as input to a system that contains the evaluation function, and have the system produce, as output, the single robotic action which has the highest probability of achieving the desired effect, out of all the candidate robotic actions. Perhaps this system could be extended to produce a set of high POE robotic actions. Robotic actions that have low POE values could then be eliminated from the candidate list.

A major difficulty in developing such a robotic action evaluation function is that the function must be able to produce POE values for all robotic actions given a desired effect. Since the number of possible robotic actions can be extremely large, it is unreasonable to believe that all possible actions could be tested and their POE values stored. Instead, given a limited number of robotic action/desired effect pairings and their corresponding POE values (developed through experimentation in the world), the evaluation function must be able to infer POE values for untested robotic action/desired effect pairings. Thus, given a training set of robotic actions, effects and their corresponding POE values, the function must be able to recognize

relationships between symbols in the training set and exploit these relationships to infer POE values for untested situations.

This thesis describes the development of the *Associative Rule Memory* (ARM), a system that possesses the following capabilities.

1. The storage of provided robotic action/desired effect pairings with POE values.
2. The ability to extract relationships between symbols which affect POE values and use these relationships to provide predictive POE values for untested robotic action/desired effect pairings.
3. The ability to provide as output, a set of robotic actions that have a high probability of achieving a desired effect, given the desired effect as input.

Figure 2.1 presents a block diagram of the system. The individual blocks are defined as

- a database of robotic actions, effects and corresponding POE values,
- the ARM model, responsible for storing the database of robotic actions and effects and predicting POE values for untested robotic action/desired effect pairings,
- an input/output interface that receives a desired effect from a user or automatic planning system as input, and responds with a set of high POE robotic actions as output,
- a recall procedure used to search the ARM to find a set of high POE robotic actions, given a desired effect.

There are several ways that the ARM can be used to aid in planning. As discussed above, the ARM can be used to rank alternative robotic actions for a given effect to reduce the size of the planning search space. An automatic

planner can then build a set of tasks that achieve a goal using the POE value as a cost metric to determine an optimal planning path to pursue. This is similar to the use of an A^* search algorithm for planning.

The ARM can also interact with a user who is familiar with the target environment to develop a plan manually. The ARM can be used to provide alternative robotic actions to the user, or can provide the user with relationships between agents in the world that have been extracted from the database of robotic actions and effects. Using these relationships, the user can determine combinations of agents that work well together, and those that should be avoided.

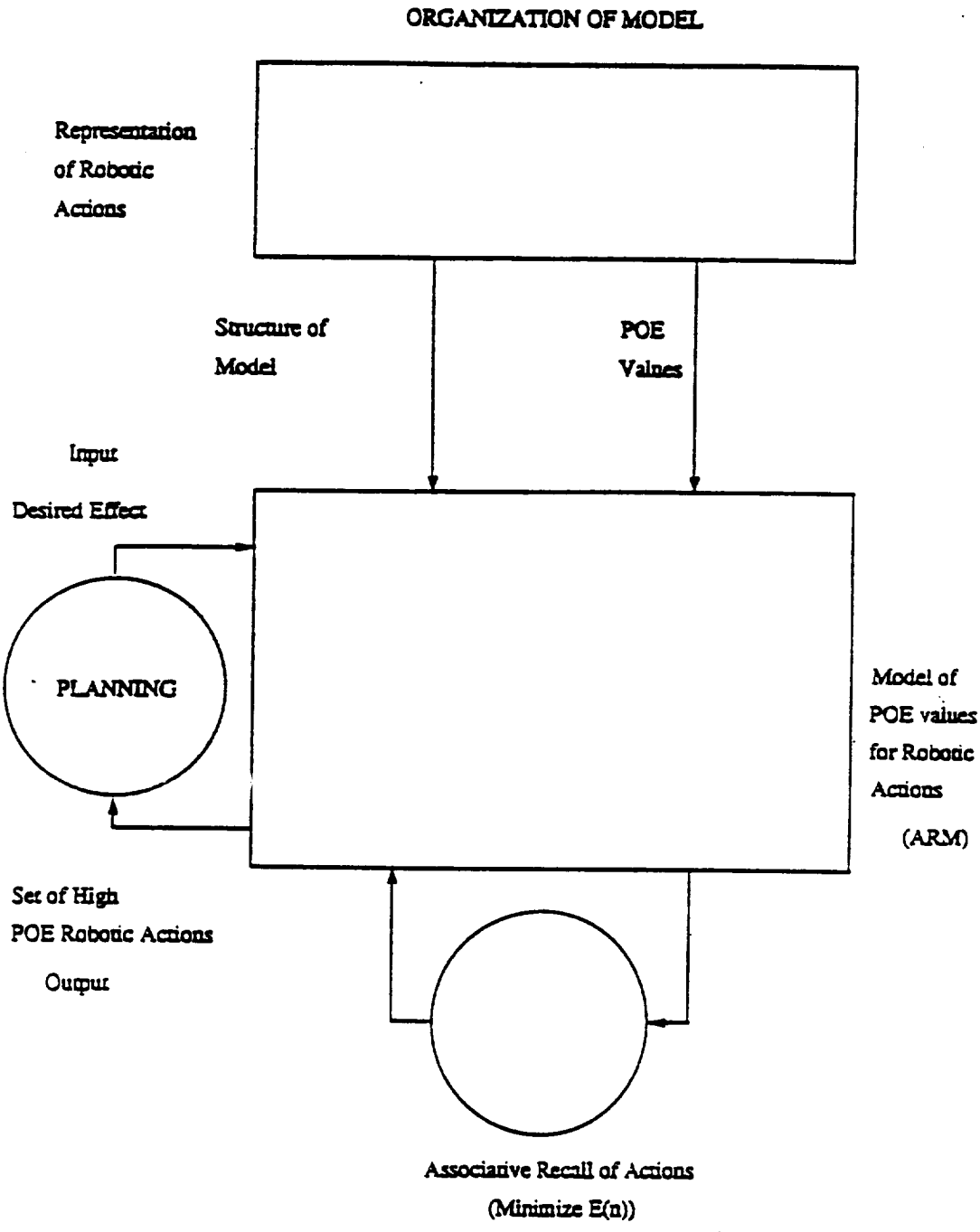


Figure 2.1: ARM system block diagram

2.4 Symbolic Structure and the ARM

The planning system can involve user interaction, or it can be an automatic planner like those previously discussed. The ARM must be able to function within such a planning system in order to be useful. This need places several structural requirements and limitations on the ARM.

Each of the previously cited planning systems represents the robotic action and effect operator as a grammar. For STRIPS and ABSTRIPS, this grammar is first-order predicate calculus. For NOAH and TIPS, a grammar is developed under the auspices of procedural nets, whose actions appears similar to the grammars used by expert systems such as OPS5 [19]. Both of these grammars contain variables that must be bound to symbols representing agents and actions in the real world. The grammar of the Organization level of the Intelligent Machine is an ordered list of primitive events that obey a set of compatibility constraints.

We will define a grammar for the ARM that is sufficiently general to encompass all the representations. The purpose of a grammar for the ARM is to define an expected set of inputs and outputs, and to limit the set of symbols that can be used together. The ARM grammar consists of two parts, *general rules* and *specific rules*. General rules contain variables for actor, action and object symbols and provide guidelines for allowable combinations. Specific rules are instantiations of general rules with symbols from the world model. Before the actual structure of the rules is presented, however, it is first necessary to discuss agent classes.

2.4.1 Agent classes

A robotic action generally consists of an actor, an action, and an object the action is performed on, often called the direct object. The robotic action may also consist of a set of indirect objects. Consider the following robotic action

$$FTS\ ATTACH\ ORU\ PALLET-H$$

achieving the effect

$$ORU\ IS-ATTACHED-TO\ PALLET-H \tag{2.1}$$

Here, the actor is represented by the symbol FTS, the action is represented by $ATTACH$, the direct object is ORU, and the indirect object is $PALLET1$. This action represents the Flight Telerobotic Servicer (FTS) attaching ($ATTACH$) the Orbital Replacement Unit (ORU) to the heavy Pallet ($PALLET$-H). It is likely that only dextrous manipulators can attach objects to other objects, so other types of actors, such as a Mobile Transporter, should not be used with the $ATTACH$ action.

A robotic action may also require a tool to accomplish its task. For example,

$$EVA\ ACTUATE\ ORU\ TOOLSET1$$

with effect:

$$ORU\ IS-ACTUATED \tag{2.2}$$

specifies that Toolset 1 must be used by the Extravehicular Astronaut (EVA) to actuate the Orbital Replacement Unit. Again, it is unlikely that a non-dextrous manipulator can successfully actuate an object. It is also unlikely that actuation can take place with an indirect object other than a tool.

We define an *agent class* to be a set of agents that possess similar characteristics with respect to the actions they can perform, or actions that can be performed with them. Following directly, a *symbol class* is the symbolic representation of an agent class. The name of a symbol class will be denote by $< \cdot >$. It is important to note that agents may belong to several agent classes. Therefore, the same symbol may exist in several symbol classes.

In the above examples, both FTS and EVA belong to the symbol class $< dex >$, the set of symbols for dextrous manipulators. Similarly, the symbol $TOOL1$ belongs to the symbol class $< tool >$, the set of tool symbols. A full list of

symbol classes used in this work along with the symbols that belong to each class, is presented in Appendix B.

2.4.2 General Rules

Symbol classes are used to limit the scope of a robotic action, by reducing the number of possible instantiations. This is demonstrated by the structure of the general rules.

General rules are formed by the grammar

$$ACTORC\ ACTION\ RECOBJC\ \rightarrow$$

$$OBJC\ STATE\ OBJC \tag{2.3}$$

where

- $ACTORC$ is a variable representing a symbol class containing actor symbols.
- $ACTION$ is a symbol denoting an action.
- $RECOBJC \in \{OBJC \mid OBJC\ RECOBJC\}$
- $\rightarrow$ separates the robotic action ($ACTORC\ ACTION\ RECOBJC$) from the effect ($OBJC\ STATE\ OBJC$) and means "produces the effect".
- $OBJC$ is a variable representing a symbol class containing object symbols.
- $STATE$ is a symbol denoting a state of an effect.

From the above description, the robotic action part of the rule can contain a number of object classes. It is important to note that $NULL$ is a symbol class, and is used as a placeholder when no agent or object is needed.

An example of a general rule is

$$< dex >\ ATTACH\ < obj >\ < obj >\ \rightarrow$$

$$< obj > \quad IS-ATTACHED-TO \quad < obj >$$

another example is

$$< dex > \quad ACTUATE \quad < obj > < tool > \rightarrow$$

$$< obj > \quad IS-ACTUATED \quad NULL$$

The general rules are used to dictate allowable combinations of symbols that the ARM should expect. They are similar to the non-instantiated rules that are present in the planning systems discussed above, and the compatability constraints of the Organization level. If a symbol combination violates a general rule, it is illegal. The ARM should be able to identify illegal symbol combinations, and prevent them from being used in planning.

2.4.3 Specific Rules

Specific rules are instantiations of general rules with agent symbols. The form of a specific rule is

$$ACTOR \; ACTION \; RECOBJ \rightarrow$$

$$OBJ \; STATE \; OBJ \tag{2.4}$$

where

- $ACTOR$ is a symbol of a symbol class representing actors.
- $ACTION$ is a symbol denoting an action.
- $RECOBJ \in \{OBJ \mid OBJ \; RECOBJ\}$.
- $\rightarrow$ has the same meaning as in general rules.
- OBJ is a symbol of a symbol class representing objects, or is $NULL$.
- $STATE$ is a symbol denoting a state of an effect.

Examples of specific rules are

$$FTS\ ATTACH\ ORU\ PALLET1\ \rightarrow$$

$$ORU\ IS-ATTACHED-TO\ PALLET1$$

and:

$$SPDM\ ACTUATE\ ORU\ TOOL1\ \rightarrow$$

$$ORU\ IS-ACTUATED\ NULL$$

For brevity, the $NULL$ symbol is sometimes omitted from the specific rule if it is known to exist in general rule to which the specific rule corresponds. For the Organization level of the Intelligent Machine, a specific rule represents an ordered string of primitive events which is an activity string.

2.4.4 Limitations of this representation

The chosen representation has several inherent limitations. These are:

1. It provides a very high level description of agents and actions and does not consider many details.

2. The symbols are discrete and cannot represent continuous states of agents (such as $SPDM\ IS-AT\ X = 123.445\ Y = 50.0$).

3. The POE value summarizes only the success or failure of a robotic action at achieving a desired effect.

4. The effect of a rule is limited to possessing three symbols:
   $(OBJ\ STATE\ OBJ)$.

Problems that need added detail or continuous states are outside the realm of this work. Since the POE value summarizes only success or failure of a robotic action, it does not detail how well the action was performed. For example, it cannot represent

a robotic action that "almost succeeds" or "nearly fails". The effect portion of the representation is small to facilitate planning from desired effects. If the effect portion were large, the system might become overwhelmed at planning possiblities using techniques like Means-End Analysis.

Further, we impose the following assumptions and constraints on our system.

1. All difficulties encountered in successfully completing a robotic task are due to interrelationships between agents and actions explicitly used to complete the task.

2. The direct object symbol in the effect of a specific rule is the same as the direct object symbol in the robotic action.

3. If the indirect object symbol in the effect of a specific rule is non-NULL, then it is the same as the first indirect object symbol in the robotic action.

4. Unexpected effects of robotic actions are not modeled by the ARM.

The first assumption states that any decrease in the POE value of a specific rule is due to interactions between symbols explicitly stated in the rule. This indicates that environmental influences of other agents not stated in the specific rule have no effect on the POE value. This is a strong assumption, but necessary if modeling of agent and action interrelationships is to take place. If this assumption was not made, unmodeled relationships could effect POE values and the ARM would not be an effective and reliable predictor.

The second and third assumptions dictate constraints on instantiations of the general rules. These assumptions are made to reduce the complexity of the ARM model. These assumptions also reduce the number of possible instantiations of a robotic action given the desired effect. We will show, however, that for many cases, the number of possible instantiations can be very large.

The fourth constraint states that only robotic actions and effect pairings that are instantiations of general rules will be considered in our system. This eliminates error states from planning and is consistent with modeling only success and failure of a robotic action by the POE value.

2.5 The ARM as a Model

The ARM is responsible for assigning POE values to robotic action/desired effect pairings. In the preceding sections, we have formalized these pairings, and named them specific rules. Also, it has been determined that the ARM should produce a set of high POE robotic actions as output when provided with a desired effect as input. Therefore, given a set of general rules, it is the responsibility of the ARM to

- assign a POE value to a specific rule that is the instantiation of a general rule,
- reject any symbol combination that is not the instantiation of a general rule,
- receive as input the effect of a specific rule, and produce as output a set of robotic actions that have a high probability of achieving the effect. It is important that the POE value for each robotic action in the set be produced as well.

As detailed above, it is not likely that the POE value for each specific rule will be available for storage. Instead, the ARM must be able to infer POE values for untested specific rules by modeling the relationships between agents and actions, and the effect these relationships have on POE values. When the ARM is presented with an untested specific rule, it can use these relationships to predict a POE value for the rule. In more general terms, the ARM must be able to model the probabilistic relationships between symbols of a known grammar.

2.5.1 Training the model

Training the ARM model develops the relationships between symbols used to predict POE values. One training technique is to have the user explicitly encode the known relationships between agents, actions and effects and store these relationships in the ARM model. This method would require a sophisticated knowledge-based system and forces the burden of relationship extraction onto the user, who must examine all tests performed in the robotic environment to create a predictive model.

A more desirable technique is to present the ARM with a set of tests that have been performed in the robotic environment, and have the ARM implicitly extract the relationships between symbols. The set of tests, called the training set, can be presented to the ARM in the form of specific rules. Each specific rule in the training set must possess a POE value that indicates the probability that the robotic action of the specific rule achieves the desired effect. It is the responsibility of the ARM to extract relationships between symbols in a specific rule and determine how the relationships effect the POE value for the rule. These predictive values can then be applied to untested specific rules that share some of the same relationships.

Using this technique, a *training set* is a collection of specific rules that have been tested in the robotic environment, along with corresponding POE values that dictate the probability that the robotic action of each specific rule achieved the desired effect of the specific rule. Together, the training set of tested specific rules and the set of general rules form a database that is provided to the ARM. This database allows the ARM to develop predictive abilities and eliminate illegal symbol combinations. The database block relative to the overall system is presented in Figure 2.2.

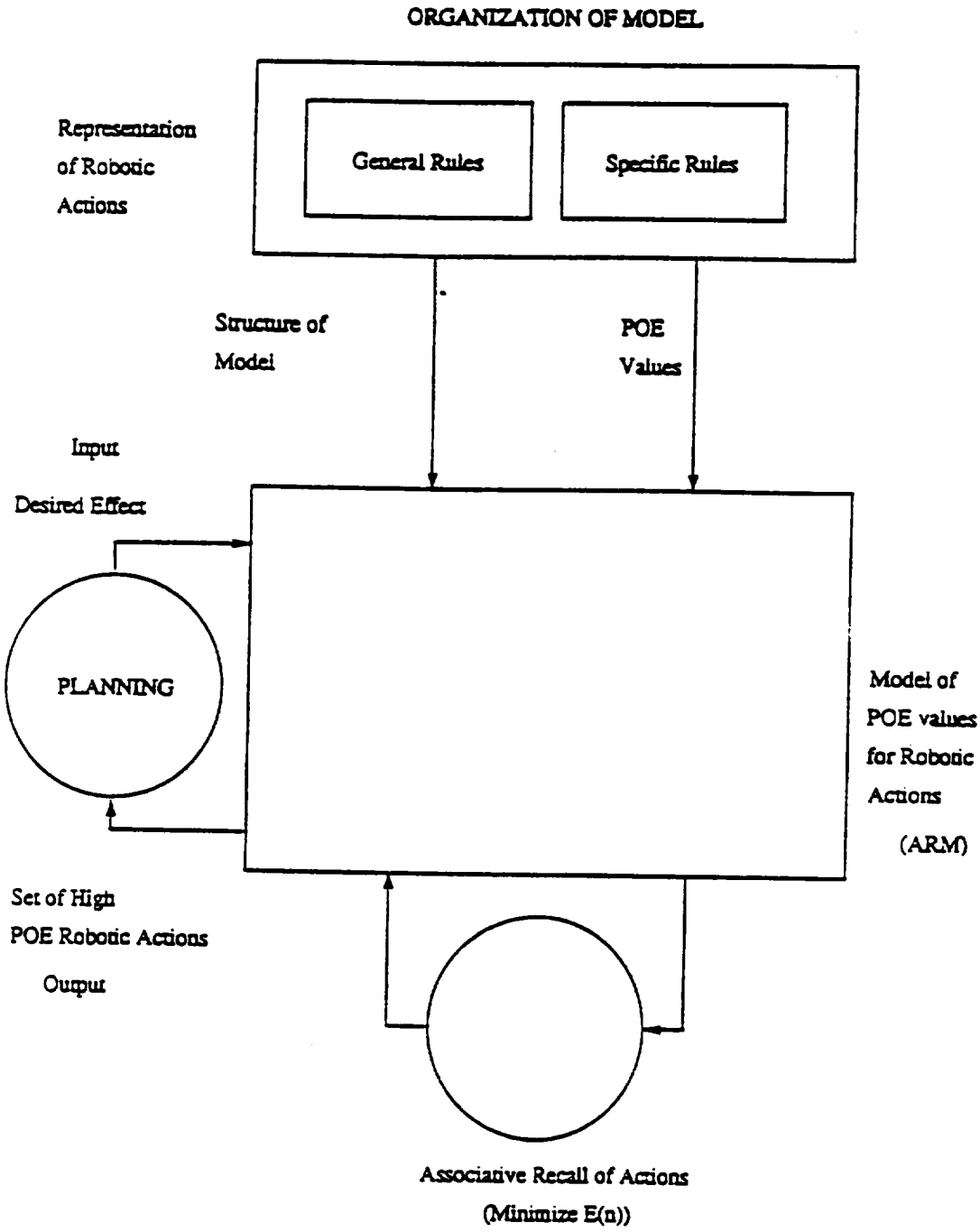


Figure 2.2: ARM system block diagram with rule database

2.5.2 The ARM as a neural network

One class of models that is particularly adept at developing implicit relationships between symbols in a training set is the artificial neural network (ANN). Each individual ANN model, however, has its strengths and weaknesses and these must be evaluated to arrive at a suitable representation. Excellent discussions comparing different ANN models are presented in [20, 21, 22, 23, 24]. The four models we shall consider are Backpropagation networks, Grossberg ART networks, Hopfield Networks, and Boltzmann Machines.

2.5.2.1 Backpropagation networks

Backpropagation networks [25] are comprised of sequential layers of simple processing units that are arranged so that the output of one layer feeds into the input of the next layer. Typically, these networks are used to learn arbitrary mappings from an input data set to a target data set. The underlying function of the processing units and connection weights in a backpropagation network is to build classification regions in the input space by creating hyperplane discriminant boundaries. In theory, a two level network can separate data into convex classification regions while a three level network can develop arbitrarily complex regions [22]. The weights are trained using a backpropagation procedure originally developed by Werbos [26]. Recently, Williams [27] developed a new procedure for training backpropagation networks using reinforcement learning.

When an input is provided, the trained network determines the boundaries within which the input falls, and produces a output corresponding to that region. Inputs that fall between the taught regions produce an output that is a blend of the output of nearby classes. In this way, the output of the backpropagation network can be somewhat continuous for nearby input values.

Applications range from classification of input data [28, 29, 30], to modeling

of transfer functions for unknown plants [31, 32, 33], to topological transformations [34]. Miyamoto et al. [35, 36] use backpropagation networks to learn coefficients of the nonlinear inverse dynamics equations for a six degree of freedom manipulator. Goldberg and Pearlmutter [37] supply a backpropagation network with window of trajectory data to learn the inverse dynamics of a two degree of freedom direct drive arm.

Day proposes a method for building an architecture in which connectionist and standard symbolic AI implementation techniques complement each other [38]. The system allows a connectionist network to observe the internal workings of a symbolic AI program and thereby learn to carry out the same problem solving behavior. Day proposes the use of a Backpropagation network to learn the AI rules. As he states, a major problem with this proposal is how to achieve the desired linkage between the two systems, so the network can observe the behavior of the rules. He does recommend the network learning be achieved by watching pre- and post-effects of the expert system chaining, where the pre-effects are the input and post-effects are the desired output of the network. In this paper, Day develops a rough architecture for this theory.

2.5.2.2 ART networks

The ART architecture, developed by Grossberg [39, 40, 21] is a biologically motivated dynamic network that is adept at online learning of pattern classifications and pattern completion. The model, as presented in Figure 2.3 contains a top down classification memory (F2) and a bottom up pattern memory (F1) as part of its attentional subsystem. Together, the two memories "resonate" as described by nonlinear gated differential equations. In steady state, memory F1 will contain a correct and complete input pattern and F2 will contain the classification of the input pattern. The weighted connections between F1 and F2 can be modified online

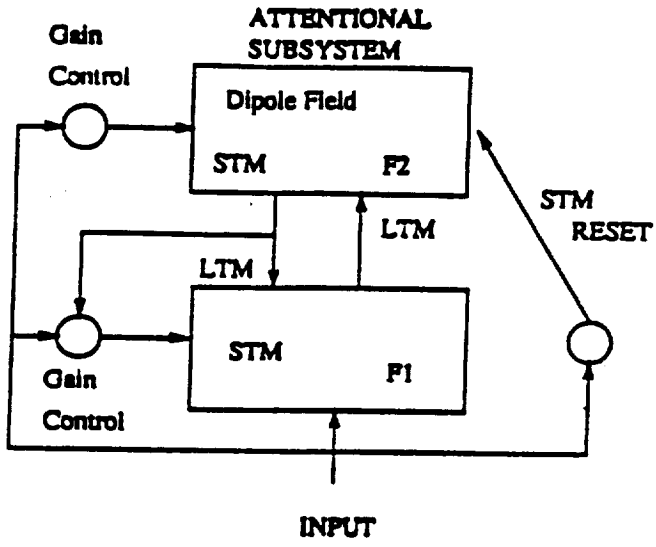


Figure 2.3: ART architecture

to assign classifications to input patterns. Some applications of this system are: modeling the timing circuit of the brain for temporal discrimination during associative learning of rewards and punishment [41]; modeling the effects of frontal lobe damage on the classification of objects by their features [42]; and alphabet learning [43].

2.5.2.3 Hopfield Networks

Hopfield networks [44, 45] are most applicable to associative memory or optimization tasks. The dynamics of the Hopfield network operate to minimize the Energy of the network, which is a function of the weights and the nodes of the network. The Energy is given by

$$E(\mathrm{N}) = \frac{1}{2} \sum_{i} \sum_{j} w_{ij} n_i n_j \tag{2.5}$$

where the state of node i is denoted n_i, $n_i \in \{-1, 1\}$, $w_{ij} \in \Re$ is a real valued weight connecting nodes i and j and $\mathrm{N} = (n_1, n_2, \cdots, n_k)$, the state of a k node

network. The network uses a gradient descent technique to alter the state of the nodes in order to find a local minimum Energy state, which is the associative recall, or optimal value of the network.

Associative recall capabilities are discussed in [44, 45] The development of stable memories, which are called terminal attractors, is discussed by Zak [46] and Hirsch [47]. It is shown in both [48] and [49] that the number of stable memories that can be stored in a Hopfield network of n nodes is $0.15n$.

For optimization, Hopfield and Tank have applied these networks to the Traveling Salesman Problem [50], where city distances and other constraints are formulated into an Energy equation, from which network weights are assigned. Using the optimization model, Touretzky [51] has developed the DUCS architecture that provides multi-level distributed representations for frame-like concept structures. The goal of this research is to develop a powerful short term memory that can construct and manipulate concepts rapidly. Given some slot name/slot filler values as cues, DUCS can retrieve entire frames from concept memory. DUCS can also complete frames that have empty slot values.

For example, given the frame

AGENT:	JOHN
VERB:	THROW
OBJECT:	x
DESTINATION:	FOX
LOCATION:	HOUSE

DUCS would retrieve the correct frame with x = ROCK.

All concepts are stored a priori by the user by fixing the connection weights. Once these weights are assigned they are fixed and the network cannot learn new concepts. This method suffers due to memory storage limitations in the Hopfield network and because it sometimes recalls incorrect frames, a residual effect of the underlying optimization technique. Hinton [52] has developed similar methods for learning concepts.

Dolan and Dyer [53] presented the CRAM system which also performs role binding in knowledge frames. The procedural memory is composed of many winner-take all cliques. Although they propose frame learning, they do not present a technique for implementing it.

2.5.2.4 Boltzmann Machines

The Boltzmann Machine [54, 55, 56] is similar to the Hopfield network in its use of an Energy function to associate an input pattern with an output pattern. The Boltzmann Machine Energy function is given by

$$E(\mathbf{N}) = \frac{1}{2}\sum_i \sum_j w_{ij} n_i n_j + \sum_j \theta_j n_j \tag{2.6}$$

where $\theta_j \in \Re$ is a bias term on a node being active, and $w_{ij} \in \Re$. Depending on implementation, $n_i \in \{-1, 1\}$ or $n_i \in \{0, 1\}$.

The Boltzmann Machine derives its name from the relationship it maintains between the Energy of the network state, and the probability the network settles in that state through Simulated Annealing, which is given by a Boltzmann distribution. Briefly put, the probability of a network state is inversely related to the Energy of that state. Simulated Annealing [57] is used as a descent technique to find the set of asserted nodes that minimizes the Energy (maximizes probability). These nodes correspond to the correct output of the network for the provided input.

Unlike Hopfield networks, Boltzmann machines also contain "hidden" nodes that are used to represent higher order relationships between input and output nodes. The purpose of these hidden nodes is similar to the middle layer nodes in a backpropagation network. Also, like backpropagation networks, Boltzmann machines can be trained on test data to associate input and output values.

Boltzmann Machines have been used for figure-ground separation in computer vision [58], combinatorial optimization [59], problems and knowledge frame recall [60]. Pearl [61, 62] has shown equivalence between belief networks and Boltzmann Machines. However, nodes in belief networks must represent propositions, and do not represent individual variables [63].

Touretzky and Hinton [64] use a distributed Boltzmann Machine architecture to represent two types of production systems. The first system contains rules that consist of pairs of working memory triples for the rule condition, and an arbitrary set of triples that must be added to or deleted from working memory as the rule effect. Typical rules are of the form

Rule-1: (F A A) (F B B) $\Rightarrow$ +(G A B) -(F A A) -(F B B)

The second production system is similar, but allows variable matching in the condition part of a rule. For example,

Rule-2: (x A B) (x C D) $\Rightarrow$ +(P D Q) -(R S T)

where x is a variable to be matched by working memory elements.

The weights of the network are fixed by the user. These weights are used to represent the rules and working memory elements. Good results are obtained with a working memory alphabet size of 25 symbols, a set of about six rules, and six elements in working memory at a time.

2.5.3 Choice of an ANN model for the ARM

Based on the various ANN models presented above, the Boltzmann Machine seems to come closest to the capabilities required by the ARM. The Boltzmann Machine allows for higher order relationships between input and output pairs, which is necessary for the modeling of robotic actions and effects. Using a Boltzmann Machine, it should be possible to associate effects of specific rules (input) with their actions (output). Also, it should be possible to store the POE of a specific rule as a function of the Energy of the network when the specific rule is asserted on the network nodes. Searching the network for a minimum Energy state using Simulated Annealing, or another optimization technique can be used to produce the robotic action that has the highest probability of achieving a given desired effect. Finally, if a suitable architecture is chosen, the general rules could also be stored.

2.6 Recall of Robotic Actions

The ARM must be able to recall a high POE robotic action given a desired effect as input. Recall requires performing a search on the ARM model. Chapter 4 will discuss how the search of the ARM model involves the minimization of a highly nonlinear, discrete function. This precludes the use of linear optimization techniques such as gradient descent, conjugate gradient, linear programming, and others [65].

Three techniques that are useful for optimizing nonlinear, discrete functions are Random Search, Simulated Annealing, and the Genetic Algorithm. Each will be quickly reviewed here, with a more detailed presentation given in Chapter 4.

Random Search [66, 67] selects random members from the space of candidate solutions, updating the current best choice whenever a member is found that is better than all previous members. This technique has been shown to converge in probability to the optimum of a given cost function over a solution space. Random Search can be quite slow, however, since it does not exploit inherent knowledge of the structure of the solution space.

Simulated Annealing [57] is similar to gradient descent, though it allows occasional uphill steps in the cost function. The uphill steps allow the search to avoid entrapment in local extrema. Simulated Annealing has been widely used with Boltzmann Machines [55, 58, 59] due to the nature of the machine's cost function. Recently, Simulated Annealing has been parallelized to reduce search time. Examples of parallel Simulated Annealing are presented in [59, 68].

Simulated Annealing can proceed quite slowly if convergence to the cost optimum is required [69]. No convergence proof has been given for parallel Simulated Annealing. Also, like Random Search, Simulated Annealing does not possess or develop any knowledge of the underlying structure of the cost function that is being optimized.

The Genetic Algorithm (GA) [70] is a third optimization technique suitable to highly nonlinear, discrete search spaces. Unlike Random Search and Simulated Annealing, the GA maintains a population of search points. In most GA applications, the search point is represented by a binary string. Using a genetic analogy, new search points are created through the selection and combination of current population members. The GA has been shown to promote high performing, short-order substrings, called "schema." The schema develop an implicit representation of the

problem space while searching for the optimum value. An excellent discussion and review of Genetic Algorithms is presented in [71].

Genetic algorithms have been used in many optimization tasks. Original work on function optimization was done by DeJong [72]. Davis has used the GA for both job shop scheduling problems [73] as well as graph coloring problems [74]. Glover [75] uses the GA to optimize the configuration of a computer keyboard.

A large amount of research has been done to make the GA more efficient. Baker [76] discusses population sampling techniques and develops one that is shown to have no bias and minimum spread. Grefenstette and Baker [77] examine the effect of fitness assignment on sampling rate in view of implicit parallelism and are able to generalize the Schema Theorem [70] to other fitness assignment functions. Goldberg [78] examines optimal sizing of GA populations for parallel and serial populations. In this work, Goldberg develops a figure of merit that describes the amount of useful schema processing in a GA. Eshelman et al. [79] examine positional and distributional biases for different crossover techniques. Many researchers have experimented with parallel GAs [80, 81, 82, 83, 84, 85].

The strength of the GA is that it develops an implicit understanding of the underlying structure of a given cost function through the propagation of short-order schema. By using semantic encodings of symbols into short binary strings, it is possible to accelerate the search procedure for a given problem [86]. Unfortunately, the GA often prematurely converges to a local extremea of a cost function. This will be further discussed in Chapter 4.

2.7 Conclusions

This chapter detailed the target world for our system, which is composed of many robotic agents and actions. It was stated that in our target world, many robotic actions can achieve the same effect. As shown by the review of automatic

planning systems, this redundancy leads to difficulty in planning, due to the size of the search space of possible actions.

To reduce the search space, an evaluation function called the Associative Rule Memory is proposed, that stores and predicts the probability that a robotic action achieves a desired effect. A grammar is provided to interface the ARM with interactive users or automatic planning systems, and an implicit training methodology is recommended.

Different artificial neural network models have been as a possible basis for the ARM. The Boltzmann Machine was determined to satisfy most of our needs, but it must be specialized to fit the ARM. To allow the ARM to produce a high POE robotic action, given a desired effect, optimization techniques were examined. The common characteristic of these techniques was the ability to optimize a highly nonlinear, discrete function.

CHAPTER 3
DESIGN OF THE ASSOCIATIVE RULE MEMORY

The previous chapter described the structure of the specific and general rules that are used in planning. The general rules and training set combine to form a database of knowledge that describe robotic actions that are possible, and actions that have been explicitly tested. Since the number of possible robotic actions may be quite large in an environment with many agents, it is reasonable to assume that the specific rules in this database encompass only a small percentage of possible actions. Since the database by itself is unable to provide probability of effect values for untested specific rules, it is only effective in planning sequences in which all required actions are present in the tested specific rules of the training set. This forms a rather small subset of possible plans.

It would be extremely useful if a mechanism or model existed to examine the tested specific rules and the general rules and use the probability of effect values to infer the agents that perform well together, and agents that do not. Such a mechanism could be used to determine probability of effect values for an untested specific rule by examining the set of agents present in the rule. Agents that together perform poorly would subtract from the POE value while other, more compatible agents would add to the POE. If we combine these POE changes with a default probability value (which represents the nominal probability of an untested specific rule), the POE for a particular untested specific rule could be determined.

Given that we know POE values for specific rules in the training set, and can somehow infer POE values for untested ones, it would greatly aid the planning process if the model could be provided with a desired effect as input and produce as output a set of robotic actions that have a high probability of achieving this desired effect along with the POE values. The user or automatic planner could then use the

model interactively to construct a sequence of highly probable robotic actions that achieve a final goal.

For example, providing the model with

$$PALLET-L\ IS-ATTACHED-TO\ TRUSS$$

the model should respond with

$$FTS\ FIXTURE\ PALLET-L\ TRUSS\quad POE: 0.97$$

$$SPDM\ ATTACH\ PALLET-L\ TRUSS\quad POE: 0.95$$

if the specific rules

$$FTS\ FIXTURE\ PALLET-L\ TRUSS\ \rightarrow$$

$$PALLET-L\ IS-ATTACHED-TO\ TRUSS\quad POE: 0.97$$

and

$$SPDM\ ATTACH\ PALLET-L\ TRUSS\ \rightarrow$$

$$PALLET-L\ IS-ATTACHED-TO\ TRUSS\quad POE: 0.95$$

exist in the model and possess the highest POE values. It is important that the technique used by the model be efficient in finding these sets of actions, since the search space of possible actions may be quite large.

Given these requirements, this chapter describes the design of the Associative Rule Memory. The function of the ARM is to:

1. Store tested specific rules and POE values.

2. Store general rules.

3. Provide predictive values for untested specific rules.

4. Given a desired effect as input, produce as output the robotic action that has the highest POE value for the effect, or a set of robotic actions, each possessing a high POE value for the effect.

This chapter focuses on items 1-3 in the above list, and forms the Associative Rule Memory block in Figure 2.1. while item 4 is examined in depth in Chapter 4. The outline of the chapter is as follows. Section 3.1 details the Boltzmann Machine, which is the neural network model used by the ARM. Section 3.2 maps the architecture of the ARM onto a Boltzmann Machine and examines the issues of input/output, representation of specific and general rules, and generalization/prediction. Section 3.3 develops a technique for training the ARM that is guaranteed to find the optimal set of weights for the given training set. Comparison is made to techniques used by other researchers for training Boltzmann Machines. Also, representation of higher order relationships between agents is discussed, along with a retraining procedure to encompass these relationships in the ARM. Examples and results of training using this procedure are presented in section 3.4. Section 3.5 examines the use of the ARM for prediction of POE values for untested specific rules. In section 3.6, extensions to the basic ARM model are presented. Section 3.7 summarizes the model and concludes this chapter.

3.1 A Description of the Boltzmann Machine Model

Before it is possible to understand how the ARM can be modeled by a Boltzmann Machine, it is necessary to develop a clear understanding of the Boltzmann Machine model. With this consideration, this section presents the fundamentals of the "generic" Boltzmann Machine model.

The Boltzmann Machine, as discussed in the previous chapter, is a constraint satisfaction network that is capable of learning underlying constraints that characterize a domain by simply being shown examples from the domain. The Machine,

a network of nodes and connections, builds an internal model of the domain by modifying the connection values, or weights in accordance with the examples it is presented.

Let the nodes of the generic Machine take values $n_i \in \{0, 1\}$. The weighted connection between a node pair represents a weak constraint between the nodes, given by a real-valued number $w_{ij} \in \Re$. As the weight increases in value between two nodes, the nodes tend to inhibit each other, i.e. both nodes will tend not to be of value 1 at the same time. This relationship is mathematically given by an energy equation, that represents the total amount of inhibition in the network for a given configuration

$$E(\mathrm{N}) = \frac{1}{2} \sum_i \sum_j w_{ij} n_i n_j + \sum_j \theta_j n_j \tag{3.1}$$

As this equation reflects, each node also has a bias associated with it (θ_j) that encourages or discourages a node from assuming the value 1. The equation also demonstrates the idea of inhibition: as more inhibitory node pairs are asserted, the energy value $E(\mathrm{N})$ increases. In effect, the inhibitions form a set of weak constraints between pairs of asserted nodes and the energy value dictates the amount of constraint violation. It is important to note that the weights between nodes are bidirectional, i.e., $w_{ij} = w_{ji}$.

The nodes of the network are divided into three sets: Input nodes, Output nodes and Hidden nodes. After fixing the node values for the input nodes, the function of the network is to find the state of the hidden and output nodes that minimizes the total inhibition, or energy of the system. The output node value at minimum energy state is the best associative recall for the given input.

The hidden nodes of the network are used to represent complex, higher order relationships between input and output nodes. Hidden nodes are required for the network to learn such functions as XOR or PARITY, which cannot be represented

by simple first order connections between input and output nodes.

The generic Boltzmann Machine uses Simulated Annealing as the optimization technique for finding the minimum energy state of the Machine. The actual Simulated Annealing algorithm is presented in the next chapter. It is sufficient to detail here, that the probability of finding the system in any global state after annealing obeys a Boltzmann distribution. In other words, after annealing, the relative probability of two global states is given by

$$\frac{P(\mathbf{N}_\gamma)}{P(\mathbf{N}_\beta)} = e^{(E(\mathbf{N}_\beta)-E(\mathbf{N}_\gamma))} \tag{3.2}$$

where $\mathbf{N}_\mathbf{x}$ is the state of the network given by $\mathbf{N}_\mathbf{x} = (n_1^x, n_2^x, \cdots, n_m^x)$ for an m node network.

Therefore, the probability that the network assumes a particular node configuration γ during annealing is given by

$$P(\mathbf{N}_\gamma) = e^{\alpha - E(\mathbf{N}_\gamma)} \tag{3.3}$$

where α is a probability normalizing constant.

A difficult and slow training procedure is used to develop the weights of the generic Machine. The thrust of this procedure is to equalize the probability distribution of input/output examples from the environment with the free running distribution of the network under the annealing process. This is summarized by the cost measure

$$G = \sum_\gamma P(\mathbf{V}_\gamma) ln \frac{P(\mathbf{V}_\gamma)}{P'(\mathbf{V}_\gamma)} \tag{3.4}$$

where $P(\mathbf{V}_\gamma)$ is the probability that the visible nodes (i.e., input and output nodes) are in state γ when their states are determined by the input/output pairings and $P'(\mathbf{V}_\gamma)$ is the corresponding probability when the network is free running under

Simulated Annealing. The weights can be changed to minimize the cost measure along the gradient

$$\frac{\partial G}{\partial w_{ij}} = \epsilon(p_{ij} - p'_{ij}) \tag{3.5}$$

where p_{ij} is the average probability of two nodes both being in state 1 when the state of the visible (input/output) units are fixed by the training set data, and p'_{ij} is the corresponding probability when the visible nodes are allowed to be changed by the Annealing process.

This gradient descent technique does not make any assumptions about the use of the hidden nodes, and allows the network to develop its own internal representation. The technique also does not restrict values of input/output pairings. Therefore, the training technique is very general.

However, these two features also make this training technique difficult and slow. At each gradient step, the algorithm requires an annealing process to determine the probability of nodes being asserted when the network is free running. This takes time. Also, the gradient technique on an arbitrary network may allow the weights to settle into a local minimum representation, in which the gradient is 0 but the error is still significant. If this occurs, retraining is necessary to obtain the global minimum.

Overall, the strength of the Boltzmann Machine for use in the ARM model is its ability to associate input/output pairs, and maintain an energy value relating these pairs. Using the Boltzmann distribution, it is possible to relate the energy to a POE value. Also, the Boltzmann Machine contains hidden nodes for representing higher order relationships. However, the weakness of the Boltzmann machine is that the training technique provided is slow and difficult, especially for large numbers of input/output pairings. Using the constraints of the ARM, however, it may be possible to develop a special case of this Machine that requires a simpler training

technique.

3.2 Mapping the ARM onto a Boltzmann Machine

Given the description of the generic Boltzmann Machine, it is now possible to specialize this network model to fit the design criteria for the ARM. In particular, we must detail the following features:

1. The mapping of specific rules onto nodes such that the network can receive a desired effect as input, and produce a set of robotic actions as output.
2. The relationship between connection weights and the POE value.
3. The topology of connection weights.
4. The mapping of general rules onto the network.
5. Higher order, hidden nodes.

3.2.1 Specific rules and network nodes

The previously described, the grammar of the specific rules is of the form

$$ACTOR\ ACTION\ RECOBJ \rightarrow$$

$$OBJ\ STATE\ OBJ$$

To allow the network to receive a desired effect as input, there must exist a set of nodes that can be asserted at the same time to represent each of the symbols of the desired effect. Similarly, to produce a robotic action as output, the network must possess sets of nodes that can represent each symbol of the "optimal" robotic action simultaneously.

To this end, we can design a network that contains three input levels and $m + 2$ output levels, where m is the number of object levels in the network. The input levels are

- OBJ_1
- $STATE$
- OBJ_2

In a similar fashion, the output levels are

- $ACTOR$
- $ACTION$
- OBJ_1
- OBJ_2
- $\vdots$
- OBJ_m

Each level (input or output) must contain sufficient nodes so that the level can represent any of the symbols corresponding to agents of that levels class. For example, if the world consists of four possible actors FTS, $SPDM$, $JRMS$, and MT, there must be a sufficient number of nodes in the $ACTOR$ level of the network to represent each of these four symbols. The class of allowable symbols for each object level is described by the general rules for a particular implementation.

There are two extremes to the symbol representation issue. One extreme maps each symbol to an individual node. This method creates a topology that is easy to understand; i.e., the user can determine the relationships between symbols by examining the weighted connections between their representative nodes. However, since this representation dictates one node per symbol, it can lead to large network sizes, and can therefore be expensive, both in terms of storage and computation. In the example above, four nodes would be required to represent the four different actors.

The other extreme is a distributed representation, where a symbol is represented by a pattern of activity of nodes. An example of a distributed representation is a binary encoding of symbols such that 2^n symbols can be represented in n nodes. In the example above, only 2 nodes would be required to represent the four actors. Distributed representations, therefore, are more efficient in term of storage requirements. Distributed representations can also be more fault tolerant if they sacrifice some representational capacity, as demonstrated by the Hopfield network in section 2.5.2.3. If a particular connection or node fails, it may be possible to provide an approximate representation of the symbol. On the negative side, distributed representations may lead to a network that is difficult to train and understand. By themselves, the weighted connections between nodes do not provide much structural information about the relationships between the symbols that the nodes represent.

The choice made for representation in the ARM is for a simple one-to-one symbol to node mapping. Although more expensive in terms of storage, our main concern is to extract and understand the relationships between symbols (or agents), so that these relationships can be exploited in planning. Also, since this is a software simulated network, we are less concerned with issues of fault tolerance.

Therefore, for each node level of the ARM, there is one node for the symbol of each agent that belongs to the class of that node level. An example of this is presented in Figure 3.1. The top half of the figure displays the nodes levels for the desired effect, or input nodes. The bottom half of the figure displays the node levels for the robotic action, or output of the network. The node labels are the symbols for the agents of a particular world model.

Given this network structure, a specific rule is said to be *asserted* on the network when each node in the robotic action part of the network is assigned the value 1 if it represents a symbol in the robotic action part of the rule. Also, each node in the effect part of the network is assigned the value 1 if it represents a symbol

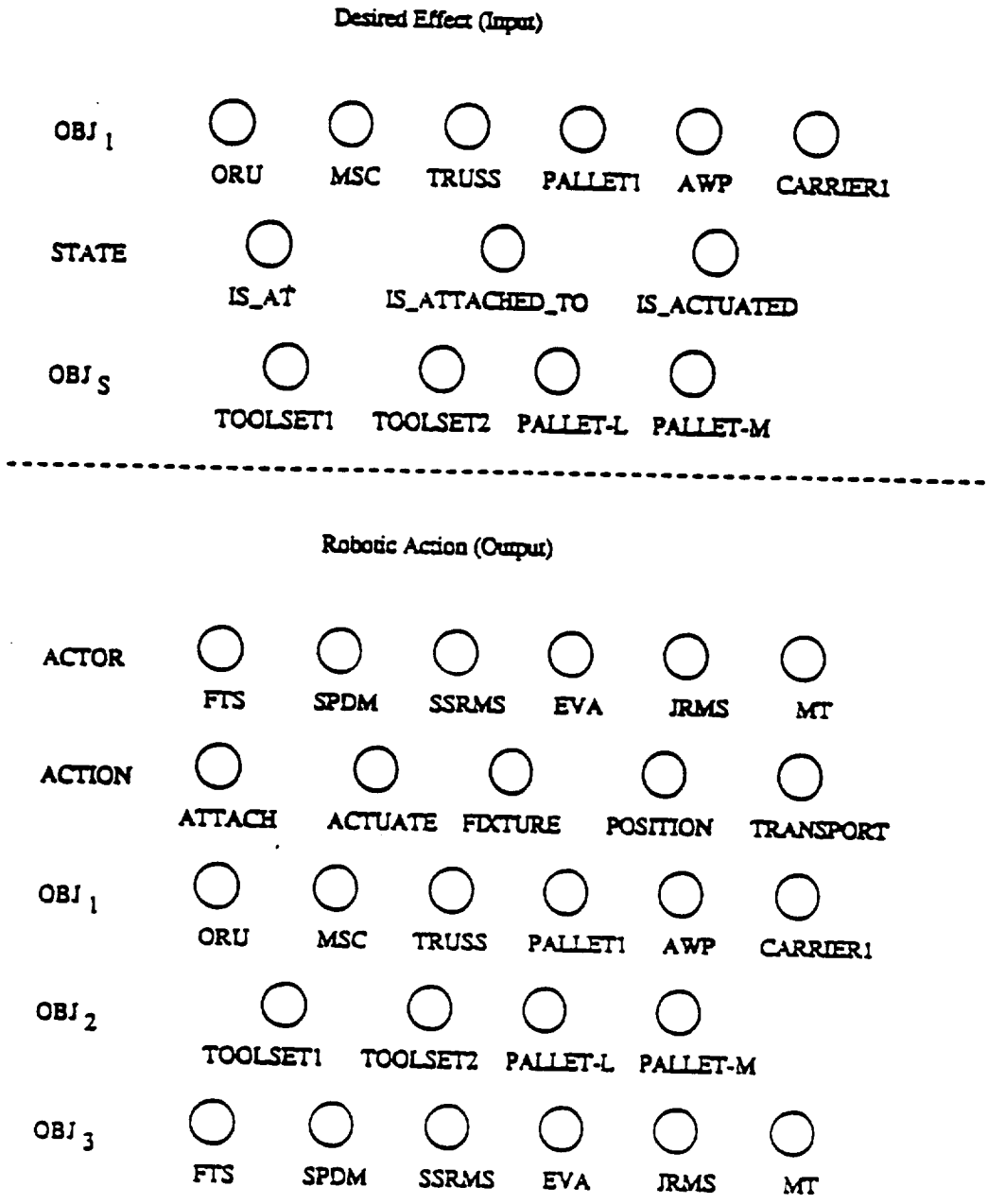


Figure 3.1: Diagram of nodes for a typical ARM network

in the effect part of the specific rule. All other nodes are assigned the value 0. For example, the specific rule

$$FTS\ ACTUATE\ TRUSS\ TOOLSET1\ JRMS \rightarrow$$

$$TRUSS\ IS-ACTUATED$$

is asserted on the network presented in Figure 3.2. The shaded nodes represent asserted, or 1 valued nodes, and the blank nodes represent unasserted, or 0 valued nodes.

3.2.2 Connection weights and the POE value

The connection weights form a set of weak constraints between pairs of asserted nodes. These weak constraints are combined into an energy function (3.1), that provides an overall indication of the amount of inhibition for a particular network configuration. A relationship exists between the energy of a configuration and the probability that the network assumes a particular configuration as shown by (3.2, 3.4) This relationship is given by a Boltzmann distribution. Using this information, we can relate the probability of effect value to the connection weights.

A simple technique for storing the POE value is to make it a function of the probability of a network configuration. In this way, the POE would be a simple function of the node configurations, and the weights of the network. A straightforward model is

$$POE(\mathrm{N}) = kP(\mathrm{N}) = ke^{\alpha - E(\mathrm{N})} \tag{3.7}$$

If we select k such that

$$k = e^{-\alpha} \tag{3.8}$$

we see that

$$POE(\mathrm{N}) = e^{-E(\mathrm{N})} \tag{3.9}$$

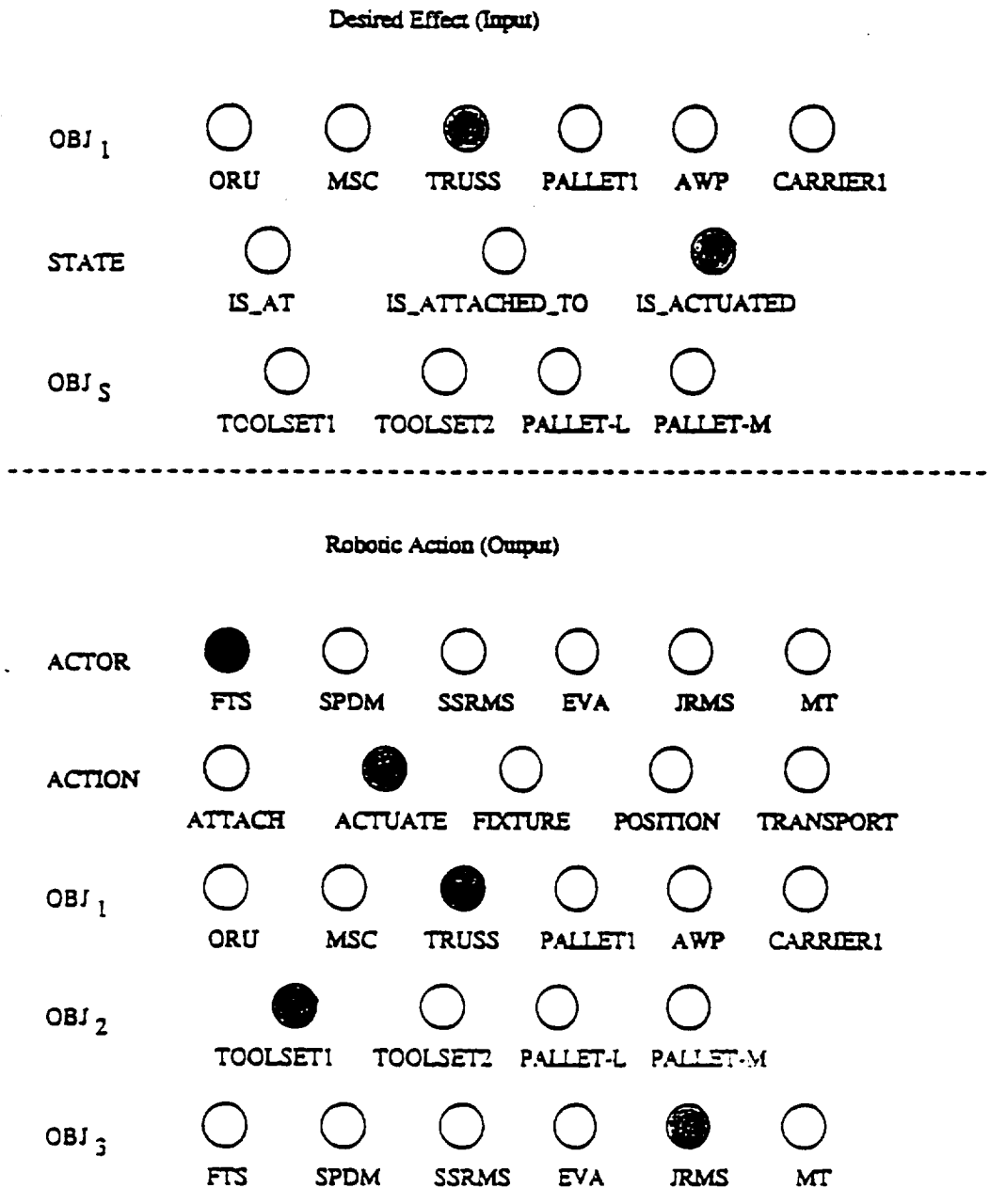


Figure 3.2: Diagram of asserted nodes for a typical ARM network

Therefore, the POE of a specific rule is exponentially related to the negative sum of the weights for the asserted nodes of the rule. We assume that the bias terms $\theta_j = 0$, since the POE value should only be affected by relationships between node pairs.

It is clear that as the weight between two nodes in a specific rule increases (increased inhibition), the POE value for the specific rule decreases. Similarly, decreasing a connection weight increases the POE value. To insure that the POE value never exceeds 1.0, we must restrict the weights of the network such that

$$(\forall i \forall j) w_{ij} \geq 0$$

This forces the energy value to be positive for all configurations and bounds the POE values by 1.0.

3.2.3 The topology of connection weights

In the generic Boltzmann Machine, all nodes are connected to each other by bidirectional weights. Due to the nature of the ARM, such a strongly connected topology is not necessary. To show this, let us restate one basic assumption that is made in section 2.4.4.

> This system assumes that any and all difficulties encountered in successfully completing a robotic task are due to interrelationships between agents and actions explicitly used to complete the task.

This assumption dictates that if a specific rule has a POE value less than 1.0 (complete certainty), then an inhibitory relationship exists between symbols in the robotic action, and symbols in the effect of that action. In other words, some subset or combination of $ACTOR$, $ACTION$, and OBJ_x symbols in the robotic action do not allow the effect of the action to occur with complete certainty. More directly, the symbols of the robotic action place constraints on the symbols representing the

effect of the action. This assumption, therefore, provides the ARM with a topological connection constraint.

The assumption dictates that inhibition is produced when a robotic action has difficulty achieving its effect, therefore connections must exist between nodes in the robotic action part of the network, and nodes in the effect part of the network. Ignoring higher order "hidden" nodes for the moment, this indicates that connections exist between the output and input nodes of the network.

It may be argued that in some cases, certain symbols in the robotic action part of a specific rule perform poorly together, and there should also be inhibitory connections between nodes in this part of the network. This argument is faulty. In actuality, poor performance occurs when symbols in the robotic action are used together to achieve a particular effect. Although higher order nodes may be required to represent a combination of robotic action symbols, the interaction and inhibition is still between robotic action nodes and the effect nodes, and not between the robotic action nodes themselves. A more detailed discussion of higher order "hidden" nodes is presented in section 3.3.2 of this chapter. Figure 3.3 shows a sample network with weighted connections between robotic action nodes and effect nodes. The connections with larger weights indicate symbol pairs that perform poorly together, and require increased inhibition to decrease the POE value. For example, suppose we assert the specific rule $\mathbf{N}_7$

$$FTS\ ACTUATE\ TRUSS\ TOOLSET1\ JRMS \rightarrow$$

$$TRUSS\ IS-ACTUATED$$

on the network and compute the energy (3.1) assuming each node bias equals 0 ($\forall j\ \theta_j = 0$) and we find that $E(\mathbf{N}_7) = 0.340$ that gives $POE(\mathbf{N}_7) = 0.712$. This indicates that the robotic action given by the above specific rule has a 71.2 percent chance of successfully achieving the effect of the specific rule.

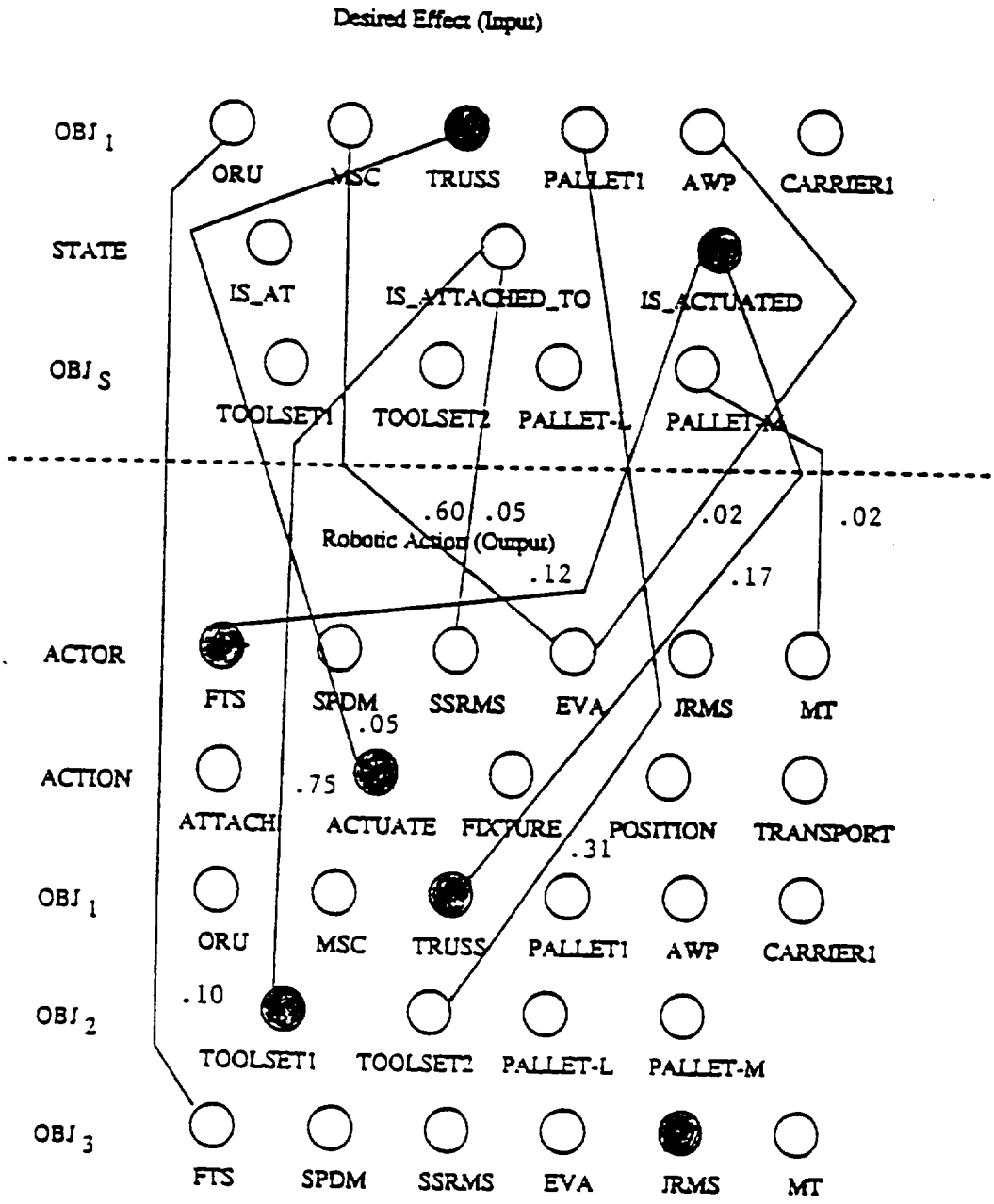


Figure 3.3: Diagram of nodes and connections for a typical ARM network

3.2.4 General rules and the ARM

As stated in the previous chapter, the purpose of the general rules is to provide a framework for possible actions. Any specific rule that is an instantiation of a general rule is a valid robotic action/effect pair. On the other hand, a specific rule is invalid if it does not fit the structure of any general rule.

The general rules should accomplish the following, when included in the ARM.

1. The ARM should produce very low POE values for any asserted specific rule that violates a general rule.

2. Given a desired effect as input, the ARM should never produce, as output, any robotic action that violates a general rule.

In fact, these constraints are strongly related. By item 1, if a set of asserted nodes violates a general rule, it should produce a low POE value. Since the ARM outputs the robotic actions with the highest POE values for a given desired effect, it should never recall one of these low POE combinations. Therefore, item 2 follows directly from item 1.

To determine how general rules should be mapped onto the network, we must examine the grammar of the rules. The description of the general rule grammar (2.3) states that each $ACTION$ symbol of a general rule is associated with a certain class of $ACTOR$ symbols and certain classes of OBJ_x symbols for each object level x. These classes represent the valid agents that can be used to perform $ACTION$ in the real world. Therefore, symbols belonging to classes that are not associated with a particular $ACTION$ symbol should be inhibited when that $ACTION$ is asserted.

Another feature of the rule grammar is that the $STATE$ symbol of a robotic effect can be caused by one or more $ACTION$ symbols. For example, the system can contain two general rules

$$< manip > \quad FIXUTRE \quad < obj > \; < obj > \; \rightarrow$$

$$< obj > \quad IS-ATTACHED-TO \quad < obj >$$

and

$$< manip > \quad ATTACH \quad < obj > \; < obj > \rightarrow$$

$$< obj > \quad IS-ATTACHED-TO \quad < obj >$$

As shown, both rules have the same $STATE$ symbol but possess different $ACTION$ symbols. Inhibition should be present between $ACTION$ symbols and $STATE$ symbols that do not occur together in general rules.

One assumption that we make is that the user will never assert a set of input nodes that cannot be generated from a robotic action. This is not a very binding constraint, but puts the burden of input consistency on the user. With the burden shifted to the user, the ARM does not need inhibition between sets of input symbol classes and the $STATE$ symbols.

The symbol constraints detailed above can be mapped directly onto the ARM network in the following manner.

1. For each $ACTION$ symbol, create a connection with large weight (inhibitory connection) between the $ACTION$ node and each $ACTOR$ and OBJ_x node whose symbol does not belong to the class of allowable symbols for that $ACTION$.

2. For each $STATE$ symbol, create a connection with large weight between the $STATE$ node and each $ACTION$ node whose symbol does not belong to the class of allowable symbols for that $STATE$.

Making these connections will inhibit asserting any $ACTION$ node that does not correspond to the $STATE$ in the input nodes, and also inhibit asserting invalid agent nodes given a valid asserted $ACTION$ node.

For example, consider the general rule

$$< manipulator > \quad ACTUATE \quad < object > \; < tool > \rightarrow$$

$$< object > \quad IS - ACTUATED \quad NULL$$

Let us assume this is the only general rule in the system that contains the action $ACTUATE$ or the state $IS - ACTUATED$. To map this onto the network in Figure 3.1, we would need the following inhibitory connections:

1. Connections between $ACTUATE$ and all $ACTOR$ nodes that are not of class $< manipulator >$.

2. Connections between $ACTUATE$ and all OBJ_1 nodes that are not of the class $< object >$.

3. Connections between $ACTUATE$ and all OBJ_2 nodes that are not of the class $< tool >$.

4. Connections between $ACTUATE$ and all OBJ_3 nodes, since the rule does not allow for more than two OBJ in the robotic action.

5. Connections between $IS - ACTUATED$ and all $ACTION$ nodes except the $ACTUATE$ node.

This is shown in Figure 3.4, where the solid lines represent inhibitory connections. Another method that achieves the same result is to initially set all connections to 1, and then remove those that are allowed by general rules.

3.2.5 Higher order nodes

Higher order, or "hidden" nodes are required by the ARM to represent relationships that cannot be mapped onto first order symbol nodes. For example, if the FTS has a difficult time achieving the state $IS - ATTACHED - TO$, this represents a first order relationship between the agent FTS and the state $IS - ATTACHED - TO$. This first order relationship is maintained in the network by a weighted connection between the two corresponding nodes. Now, consider the

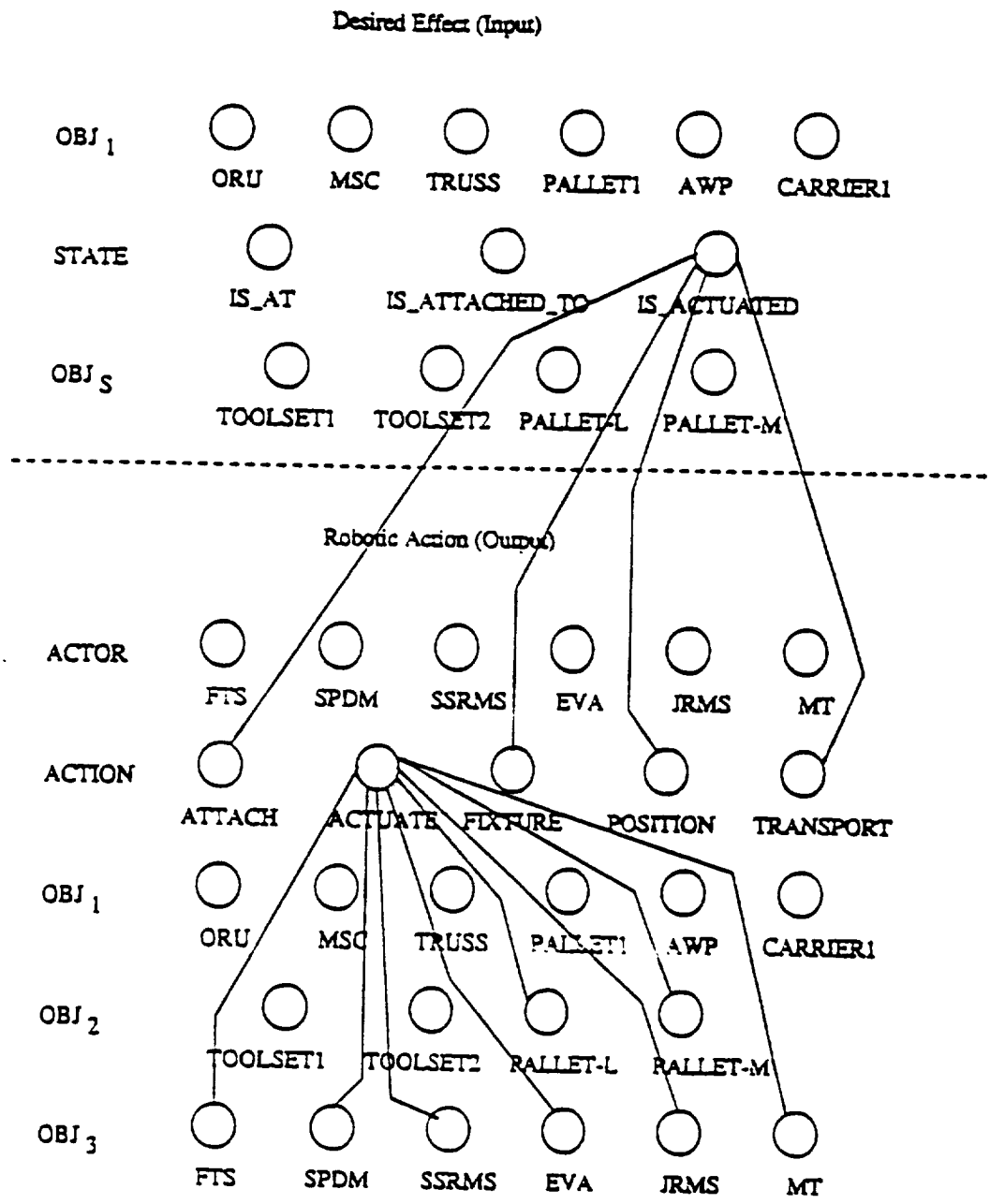


Figure 3.4: Diagram of general rule inhibitions for a typical ARM network

case of the FTS having a hard time achieving the state $IS-ATTACHED-TO$ when trying to attach to $PALLET-L$. This presents a higher order relationship where $FTS + PALLET-L$ together inhibit $IS-ATTACHED-TO$. A second order node is required to represent the combination $FTS + PALLET-L$ and a weighted connection is made from this node to the $IS-ATTACHED-TO$ node. It is possible to have third, fourth, $\cdots$, nth order nodes if necessary.

The generic Boltzmann Machine allows the provided hidden nodes to develop the required higher order relationships through its training procedure. Through this training, the hidden nodes can implicitly assume representations needed by the network to accurately represent the training set. Only a small number of hidden nodes are generally needed to develop the representation using this technique; however, the savings in storage is paid for by a very difficult and slow training that is not guaranteed to find a correct representation for the training set [55].

To insure accurate representation of the POE values for specific rules, and to simplify the training technique, the ARM uses higher order nodes that explicitly represent the combination of symbols. Symbol nodes, such as $FTS + PALLET-L$, will be added to the network by the training procedure when higher order relationships are detected. This procedure will be described later in the chapter. When computing POE values for specific rules, a higher order node in the ARM network is automatically asserted when the symbols it represents are asserted on the first order nodes. An example of higher order nodes is presented in Figure 3.5

It is important to note that the higher order nodes will only be required for symbols in the robotic action part of the specific rule. This is because it is the combination of agents in the robotic action that may have difficulty in achieving a stated effect. This method will, however, require more higher order nodes than needed by the original training technique.

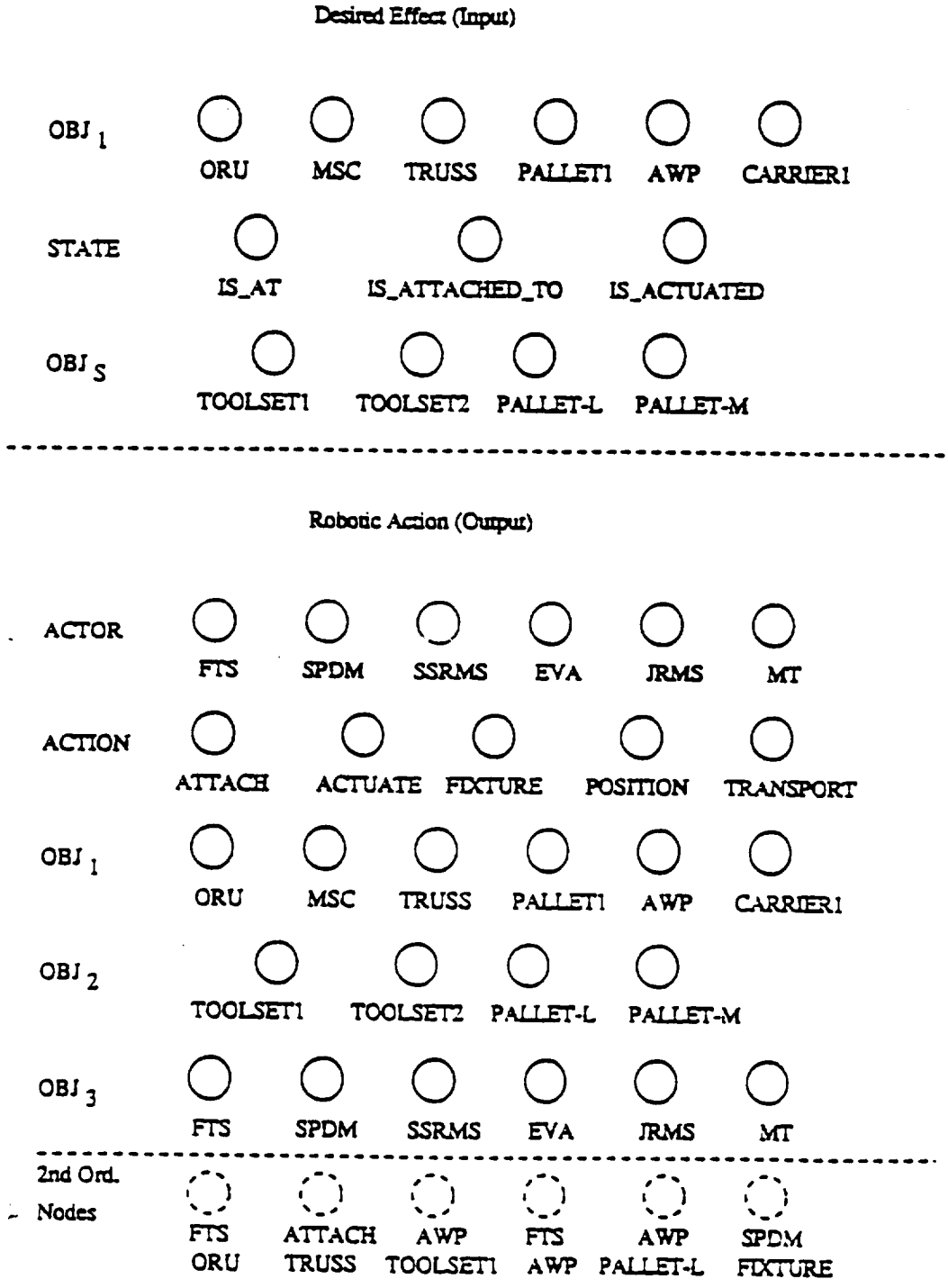


Figure 3.5: Diagram of higher order nodes for a typical ARM network

3.3 Training the ARM

The previous section gave a general sense of how weights in the network should be allocated to store the POE value. Agents that do not work well in achieving an effect should correspond to nodes that have high weight values between them. The high weight values increase the energy value for the configuration, which in turn decreases the POE value of the configuration. An algorithm must be developed to formalize this concept. In other words, given an initial training set of tested specific rules, the network must adopt weight values to reflect the performance of the agents at their task, and correctly represent their known POE values.

Training the network can be a difficult procedure. The training technique used must overcome the following problems.

1. The POE of each tested rule must be stored with a defined degree of accuracy.

2. Since tested specific rules often contain overlapping subsets of symbols, the rules also share the same connections in the ARM network. Altering one connection weight to more accurately represent the POE of a particular tested specific rule may lead to less accurate representations for other specific rules in the training set. The training technique must possess a method for altering weights that can overcome this difficulty.

3. From any initial state of the network, given a particular training set, the training algorithm must always converge to the same set of weights. This requirement stipulates that the relationships between symbols for a given training set always map to the same weight representation. Thus, the requirement prevents multiple solutions from occurring.

4. Higher order relationships between symbols must be discovered, corresponding nodes must be added, and weights must be correctly trained. If unnecessary higher order nodes are created, they must be pruned.

Since the ARM is a Boltzmann Machine specialized in both connection topology and higher order node structure, it is possible to develop a training technique that is not as complicated as the original method designed for the generic Boltzmann Machines. The technique chosen is to minimize the sum-squared distance between the energy of the ARM when a tested specific rule is asserted on the nodes and the energy value of the tested specific rule provided by the training set.
Let

- $POE_t(\mathrm{N}_\gamma)$ be the POE value for a tested specific rule γ, as provided by the training set.
- $E_t(\mathrm{N}_\gamma) = -ln(POE_t(\mathrm{N}_\gamma))$. This is the energy value to be stored in the network for specific rule γ, as given by (3.8).
- $E(\mathrm{N}_\gamma)$ be the energy of specific rule γ when asserted on the network.

For a network with only first order nodes, we know that

$$E(\mathrm{N}_\gamma) = \frac{1}{2}\sum_i \sum_j w_{ij} n_i^\gamma n_j^\gamma \tag{3.10}$$

where $\mathrm{N}_\gamma = (n_1^\gamma, n_2^\gamma, \cdots, n_m^\gamma)$ for an m node network, the state of node k for training set rule γ is $n_k^\gamma \in \{0, 1\}$, and i and j index the first order nodes in the network (assuming 0 bias).

Let us define

$$G = \sum_\gamma \{E(\mathrm{N}_\gamma) - E_t(\mathrm{N}_\gamma)\}^2 + \alpha_1 \sum_i \sum_j w_{ij}^2 \tag{3.11}$$

G is a cost function that represents the sum squared error between the asserted and desired energy value of a tested specific rule. By selecting an appropriate α_1 and minimizing G, the energy difference will approach 0 for all rules in the training set. When this occurs, the POE of all tested specific rules will be accurately represented by the network.

We can show that the function G is strictly convex. If G is strictly convex, it possesses a single minimum value; i.e., minimizing G will produce only one solution. Therefore, the relationships between symbols for a given training set will always map to the same weight representation. Also, a strictly convex function over a convex space contains no local minima except the global minimum. If G is strictly convex, a simple technique can be used to find its minimum value.

Theorem 3.1: G is strictly convex.

Proof:

Let

$$\mathbf{W} = \begin{bmatrix} w_{11} \\ w_{12} \\ \vdots \\ w_{21} \\ \vdots \\ w_{mm} \end{bmatrix} \tag{3.12}$$

This is the column vector representation of the weight matrix.

$$\mathbf{S}_\gamma = \begin{bmatrix} n_1^\gamma n_1^\gamma \\ n_1^\gamma n_2^\gamma \\ \vdots \\ n_2^\gamma n_1^\gamma \\ \vdots \\ n_m^\gamma n_m^\gamma \end{bmatrix} \tag{3.13}$$

This is the column vector representation of the node pairs for the tested specific rule γ. We can see that

$$E(\mathbf{N}_\gamma) = \frac{1}{2} \sum_i \sum_j w_{ij} n_i^\gamma n_j^\gamma = \frac{1}{2} \mathbf{W}^\mathbf{T} \mathbf{S}_\gamma \tag{3.14}$$

and we know that

$$\mathbf{W}^{\mathbf{T}}\mathbf{S}_\gamma \geq 0 \tag{3.15}$$

Therefore,

$$(\mathbf{W}^{\mathbf{T}}\mathbf{S}_\gamma)^2 = \mathbf{W}^{\mathbf{T}}\mathbf{S}_\gamma\mathbf{S}_\gamma^{\mathbf{T}}\mathbf{W} \geq 0 \tag{3.16}$$

for all $\mathbf{W}$. If we let

$$\bar{\mathbf{S}} = \sum_\gamma \mathbf{S}_\gamma\mathbf{S}_\gamma^{\mathbf{T}} \tag{3.17}$$

then

$$\sum_\gamma(\sum_i\sum_j w_{ij}n_i^\gamma n_j^\gamma)^2 = \mathbf{W}^{\mathbf{T}}\bar{\mathbf{S}}\mathbf{W} \geq 0 \tag{3.18}$$

By (3.18) the matrix $\bar{\mathbf{S}}$ is positive semi-definite over a convex space defined on $\mathbf{W}$.

If $\mathbf{W} \neq [0]$, we can see that

$$\alpha_1\mathbf{W}^{\mathbf{T}}\mathbf{W} = \mathbf{W}^{\mathbf{T}}(\alpha_1\mathbf{I})\mathbf{W} > 0 \tag{3.19}$$

where $\mathbf{I}$ is the identity matrix and $\alpha_1 > 0$. Therefore,

$$\mathbf{W}^{\mathbf{T}}(\bar{\mathbf{S}} + \alpha_1\mathbf{I})\mathbf{W} > 0 \tag{3.20}$$

which indicates that $\bar{\mathbf{S}} + \alpha_1\mathbf{I}$ is positive definite over a convex space defined on $\mathbf{W}$.

It is known that a function possessing a positive definite Hessian matrix is strictly convex. Since $\bar{\mathbf{S}} + \alpha_1\mathbf{I}$ is the Hessian of G, G is strictly convex over a convex space defined on $\mathbf{W}$. Therefore, G possesses a single minimum.

Q.E.D..

It is also known that a positive definite function possesses no local optima except for the global minimum. The minimum of G can therefore be found using a simple gradient search technique. It may be possible to use an accelerated technique such as constrained conjugate gradient to find the minimum. One must be careful, however, to insure that the accelerated technique doesn't require inversion of the $\bar{\mathbf{S}}$ matrix, since it is very large. Using an active contraint technique may work well

if combined with a unconstrained search method, since the constraint matrix may often be singular.

The gradient of G with respect to the weights can be expressed as

$$\frac{\partial G}{\partial w_{ij}} = 2\sum_{\gamma}\{E(\mathbf{N}_\gamma) - E_t(\mathbf{N}_\gamma)\}n_i^\gamma n_j^\gamma + 2\alpha_1 w_{ij} \tag{3.21}$$

subject to $w_{ij} \geq 0$. The minimum of G can be found numerically by iterating on

$$w_{ij} = wij - \epsilon\frac{\partial G}{\partial w_{ij}} \tag{3.22}$$

for all weights w_{ij}, and a small positive step size ϵ. The correct weights are found as all the partial derivatives approach 0.

3.3.1 Training higher order nodes

The α_1 term in (3.11) allows the function G to be strictly convex, instead of convex, and thereby insures a single solution. The term affects the weights by forcing weight values to be as small as possible without greatly disturbing the difference between the asserted and desired energy for a specific tested rule. We will call this term a "forcing function."

We have stated that equation (3.11) applies to training a network with only first order nodes. If second order nodes are included in training, (3.11) must be modified to

$$G = \sum_{\gamma}\{E(\mathbf{N}_\gamma) - E_t(\mathbf{N}_\gamma)\}^2 + \alpha_1\sum_i\sum_j w_{ij}^2 + \alpha_2\sum_i\sum_h w_{ih}^2 \tag{3.23}$$

where i, j denote first order nodes and h denotes second order nodes. The gradient for the second order connections is

$$\frac{\partial G}{\partial w_{kj}} = 2\sum_{\gamma}\{E(\mathbf{N}_\gamma) - E_t(\mathbf{N}_\gamma)\}n_i^\gamma n_j^\gamma + 2\alpha_2 w_{ih} \tag{3.24}$$

for second order connections. The gradient in (3.21) is still correct for connections between first order nodes.

In (3.23), the term

$$\alpha_1 \sum_i \sum_j w_{ij}^2 \tag{3.25}$$

is used to minimize the weights between first order nodes. For this term, the indices i and j index first order nodes only. Similarly, the term

$$\alpha_2 \sum_i \sum_h w_{ih}^2 \tag{3.26}$$

is used to minimize the weights between second order nodes and first order nodes. For this term, the index h represents higher order nodes. It is important to remember that the higher order nodes only appear on the robotic action side of the ARM, and connect to effect nodes on the input side of the network.

There is a purpose for separating the weight forcing function into two parts (3.25; 3.26). It is generally desirable for the network to assume that interaction between symbols is a first order relationship. Only when first order relationships cannot adopt the proper representation should higher order nodes be used. If existing higher order nodes are unnecessary, the term in (3.26) will force all of its connections to go to 0. If a higher order node exists with connection weights of 0 to each first order node, the higher order node has no effect on the energy of any specific rule. This node can be removed or "pruned" from the network without changing the representation of any rule in the training set. This reduces the number of higher order nodes required by the network, and prevents a combinatorial explosion of nodes from occurring.

To implement this strategy, α_1 should be much less than α_2. This allows weight to build up in first order connections by penalizing excessive connection weights from higher order nodes. Actual experimental values for α_1 and α_2 are presented below. Of course, if third, fourth, $\cdots$, order nodes are needed for representation, each would require a forcing function with constants

$$\alpha_1 < \alpha_2 < \alpha_3 \cdots \tag{3.27}$$

added to the cost equation G.

3.3.2 Developing higher order nodes

With a weight training technique outlined, it is now possible to discuss the development of higher order nodes in the network. Initially, the robotic action portion of the network contains only first order nodes. The network is trained using gradient descent (3.22) and eventually the magnitude of the gradient approaches 0. When the magnitude of the gradient is within an acceptable neighborhood of 0, the descent is stopped, and the value

$$G_\gamma = \{E(\mathbf{N}_\gamma) - E_t(\mathbf{N}_\gamma)\} \tag{3.28}$$

is computed for each rule γ in the training set. For each rule γ, if G_γ is within an acceptable neighborhood of 0, the training is complete, and the training set has been accurately modeled by the network. If the gradient is near 0 and G_γ is not acceptable, this indicates the network was unable to develop suitable weights for the training set. In this case, higher order relationships exist between symbols in the specific rules that the first order weights were unable to sufficiently represent.

At this point, higher order nodes must be added to the network. One inefficient method would be to add second order nodes for each pair of symbols for each specific rule in the training set and then retrain using the gradients given by (3.21) and (3.24). For large training sets, excessive computation would be required to compute all the gradients. Many of the second order nodes would be unnecessary, and all their connection weights would go to 0.

In general, a rule whose asserted POE value is much greater than its actual POE is a likely candidate for possessing a higher order relationship. If the asserted POE value is higher than the desired POE value for the rule, this indicates that the connection weights were not allowed to assume a large enough value to accurately represent the rule. This is due to the connection overlap between this rule, and other

rules that want the connection weight to assume a lower value. Additional degrees of freedom are required to build up the weight required by this rule in order to achieve a lower POE value; however, all first order relationships have been exhausted by the current network. The additional degrees of freedom for this "higher order rule" must come from connections to higher order nodes.

Often, rules in the training set that overlap first order connections with a higher order rule are not accurately represented after the initial training. Since the higher order rule tries to add extra weight to the first order connections, it tends to reduce the POE values for other rules that use this connection. Overall, therefore, the presence of higher order relationships can be detected by an "averaging" of asserted POE values over the training set.

The method adopted in this work uses the heuristic information discussed above to add higher order nodes as necessary, and to prune them after each iteration. After the initial training, the rules in the training set whose asserted POE values are much greater than their desired POE values are selected. Since rules in the training set overlap connections, these rules force other training set rules to assume bad representations. Adding second order nodes for these rules may add the required degrees of freedom to allow the network to achieve a suitable representation for all rules in the training set. The network is retrained, and the unnecessary second order nodes are discarded. If G_γ is still not acceptable for each rule in the training set, the next set of poorly represented rules are selected and the process continues. If all rules are exhausted and the representation is still not accurate, the process repeats itself with third, fourth, etc. order nodes. The algorithm proceeds as follows

Let

- Θ_1 be the gradient cutoff.
- Θ_2 be the desired accuracy of a specific rule in the training set.

- Θ_3 and Θ_4 are variables denoting limits of the POE error band for higher order node consideration.
- Θ_5 be the minimum connection weight allowed before the connection is assumed to be unnecessary.
- H_γ is the set of training set rules that contain poorly represented POE values.
- m be the number of first order nodes.
- m_h be the number of higher order nodes.
- $Differror$ be the POE error band for higher order node consideration.
- γ be a training set specific rule.

Repeat

1. For $i = 1$ to m and for $j = 1$ to m,

   (a) Compute $\frac{\partial G}{\partial w_{ij}}$

   (b) Let $w_{ij} = w_{ij} - \epsilon \frac{\partial G}{\partial w_{ij}}$

Until $\sum_i \sum_j (\frac{\partial G}{\partial w_{ij}})^2 < \Theta_1$

Let $n = 1$

While $\|G_\gamma\| > \Theta_2$ for any rule γ in the training set do

1. $n = n + 1$

2. $\Theta_3 = 1.0$

3. $\Theta_4 = Differror$

4. While $\|G_\gamma\| > \Theta_2$ for any rule γ in the training set and all training set rules not exhausted do

   (a) Let $H_\gamma = \{\gamma : \Theta_3 > POE(\mathrm{N}_\gamma) - POE_t(\mathrm{N}_\gamma) > \Theta_3 - \Theta_4\}$

   (b) For each rule $\gamma \epsilon H_\gamma$, add all order n nodes to network.

   (c) Repeat

      i. For $i = 1$ to m, and for $j = 1$ to m,

         A. Compute $\frac{\partial G}{\partial w_{ij}}$

         B. Let $w_{ij} = w_{ij} - \epsilon \frac{\partial G}{\partial w_{ij}}$

      ii. For $i = 1$ to m and for $h = 1$ to m_h,

         A. Compute $\frac{\partial G}{\partial w_{ih}}$

         B. Let $w_{ih} = w_{ih} - \epsilon \frac{\partial G}{\partial w_{ih}}$

   (d) Until $\sum_i \sum_j (\frac{\partial G}{\partial w_{ij}})^2 + \sum_i \sum_h (\frac{\partial G}{\partial w_{ih}})^2 + < \Theta_1$

(e) Remove the second order nodes that have all connection weights less than Θ_5.

(f) Compute G_γ for each rule γ in the training set.

(g) $\Theta_3 = \Theta_3 - \Theta_4$

5. end

end

Using this algorithm, the network undergoes a series of expansions and contractions until the proper nodes are added that encompass the higher order relationships present in the training set.

3.4 Some Training Examples

This section presents several sample ARM networks and demonstrates the results of the training procedure. The example evolves from simple, first order nodes in Figures 3.6 and 3.7 to second order nodes in Figures 3.8 and 3.9. In these figures, if no connection appears between robotic action and effect nodes, it indicates that the connection weight is 0.

Figure 3.6 presents a training set of three tested specific rules. As demonstrated by the training set, the robotic action

$$SSRMS\ DEPLOY\ ORU\ TOOLSET2\ \rightarrow \tag{3.29}$$

has a difficult time achieving the effect

$$ORU\ IS-DEPLOYED \tag{3.30}$$

since the POE value for this rule is only 0.30. A POE of 0.30 corresponds to an energy value of 1.20 by (3.8). The sum of the weights for this specific rule when asserted on the network, as given by (3.10), must equal 1.20 after the network has

been trained. If each specific rule in the training set is asserted one at a time on the network in Figure 3.6 and the POE of each rule is calculated, one can see that the network has achieved suitable representations.

Based on the other two rules in the test set, the training procedure allocates most of the weight to the connection between nodes $SSRMS$ (robotic action node) and ORU (effect node), and to the connection between nodes $TOOLSET2$ (robotic action node) and ORU. These inhibitory connections indicate that the $SSRMS$ has a hard time affecting the ORU and that $TOOLSET2$ also cannot affect the ORU.

Figure 3.7 adds another specific rule to the training set, and the network is retrained. The new rule indicates that $TOOLSET2$ can reliably affect the ORU. Therefore, most of the cause of the previous difficulty is attributed to the connection between nodes $SSRMS$ and ORU.

Figure 3.8 adds a rule that demonstrates that the $SSRMS$ can affect the ORU reliably. Inhibition can no longer be placed solely on first order connections, because adequate representation of all the rules in the training set would not be possible. The network deduces that a second order relationship must exist, and creates six second order nodes corresponding to the pairwise combinations of the robotic action symbols of (3.29). The network is retrained, the second order node $DEPLOY + ORU$ is found to be unnecessary, and is pruned. The remainder of the weight that cannot be assigned to first order connections is divided evenly among the connections between second order robotic action nodes and effect nodes.

In Figure 3.9, a final rule is added to the training set. During retraining, this rule causes the pruning of two more second order nodes that are deemed unnecessary. During retraining, therefore, a total of six second order nodes were added, of which three were subsequently removed.

In each of the training sets, the following constants were used.

FTS DEPLOY ORU TOOLSET1 -> ORU IS_DEPLOYED POE: 0.95
SSRMS DEPLOY ORU TOOLSET2 -> ORU IS_DEPLOYED POE: 0.30
SSRMS DEPLOY MSC TOOLSET2 -> MSC IS_DEPLOYED POE: 0.92

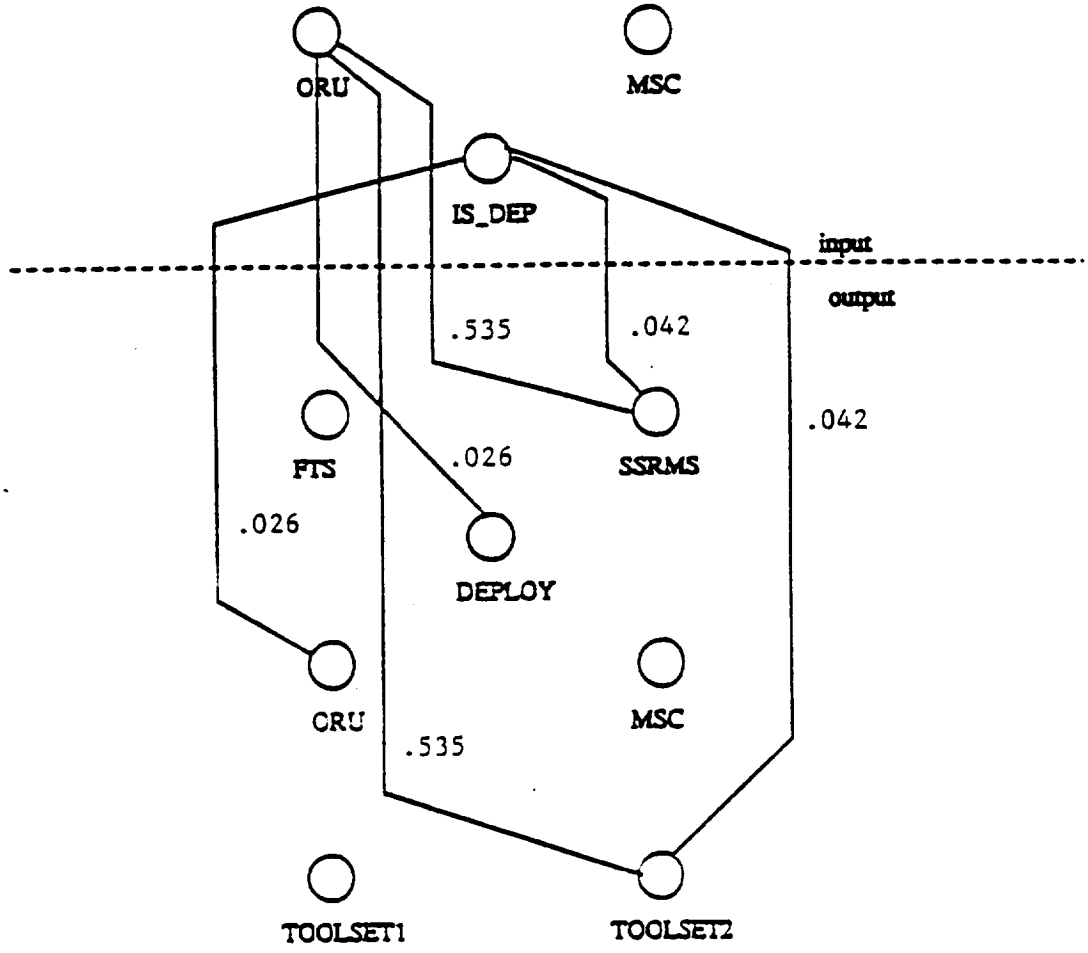


Figure 3.6: A training example

FTS DEPLOY ORU TOOLSET1 -> ORU IS_DEPLOYED POE: 0.95
SSRMS DEPLOY ORU TOOLSET2 -> ORU IS_DEPLOYED POE: 0.30
SSRMS DEPLOY MSC TOOLSET2 -> MSC IS_DEPLOYED POE: 0.92
FTS ACTUATE ORU TOOLSET2 -> ORU IS_ACTUATED POE: 0.88

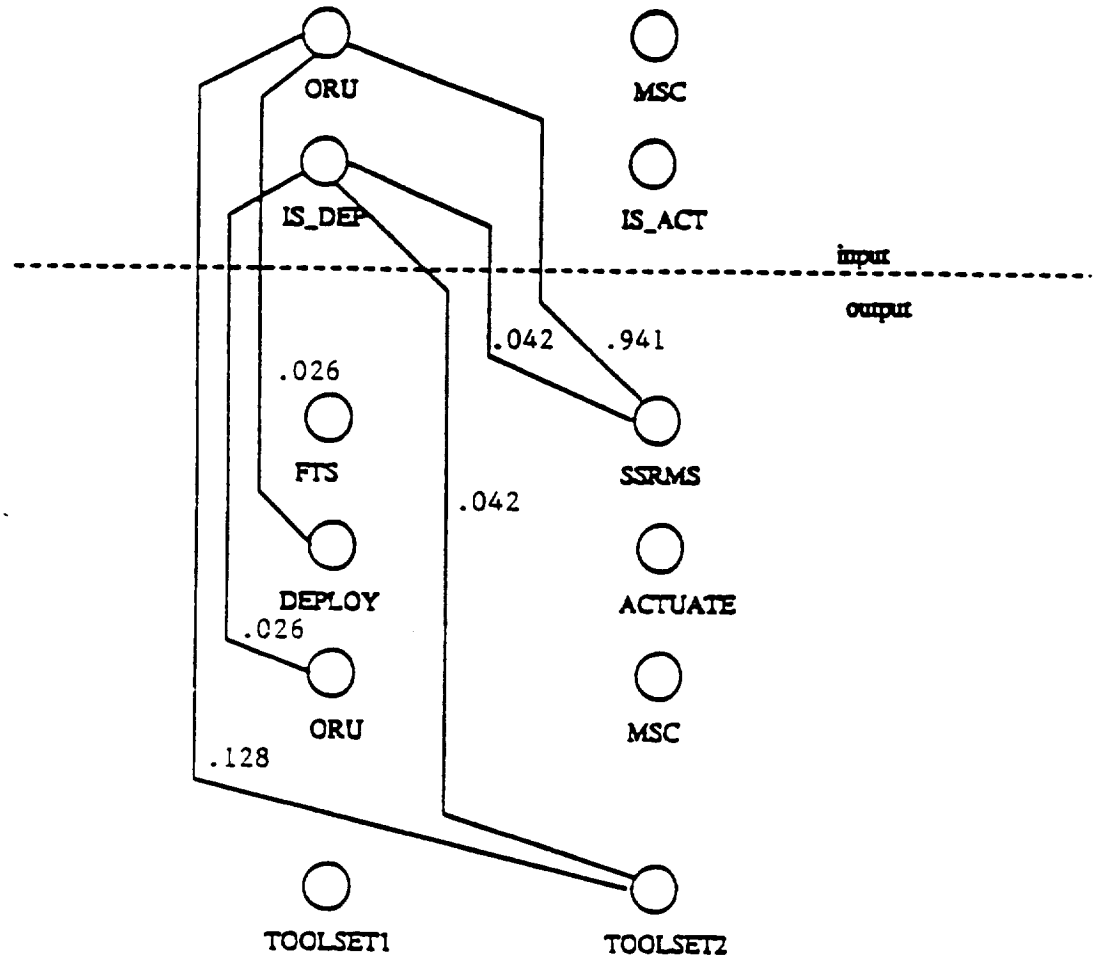


Figure 3.7: A second training example

FTS DEPLOY ORU TOOLSET1 -> ORU IS_DEPLOYED POE: 0.95
SSRMS DEPLOY ORU TOOLSET2 -> ORU IS_DEPLOYED POE: 0.30
SSRMS DEPLOY MSC TOOLSET2 -> MSC IS_DEPLOYED POE: 0.92
FTS ACTUATE ORU TOOLSET2 -> ORU IS_ACTUATED POE: 0.88
SSRMS POSITION ORU TRUSS -> ORU IS_AT TRUSS POE: 0.94

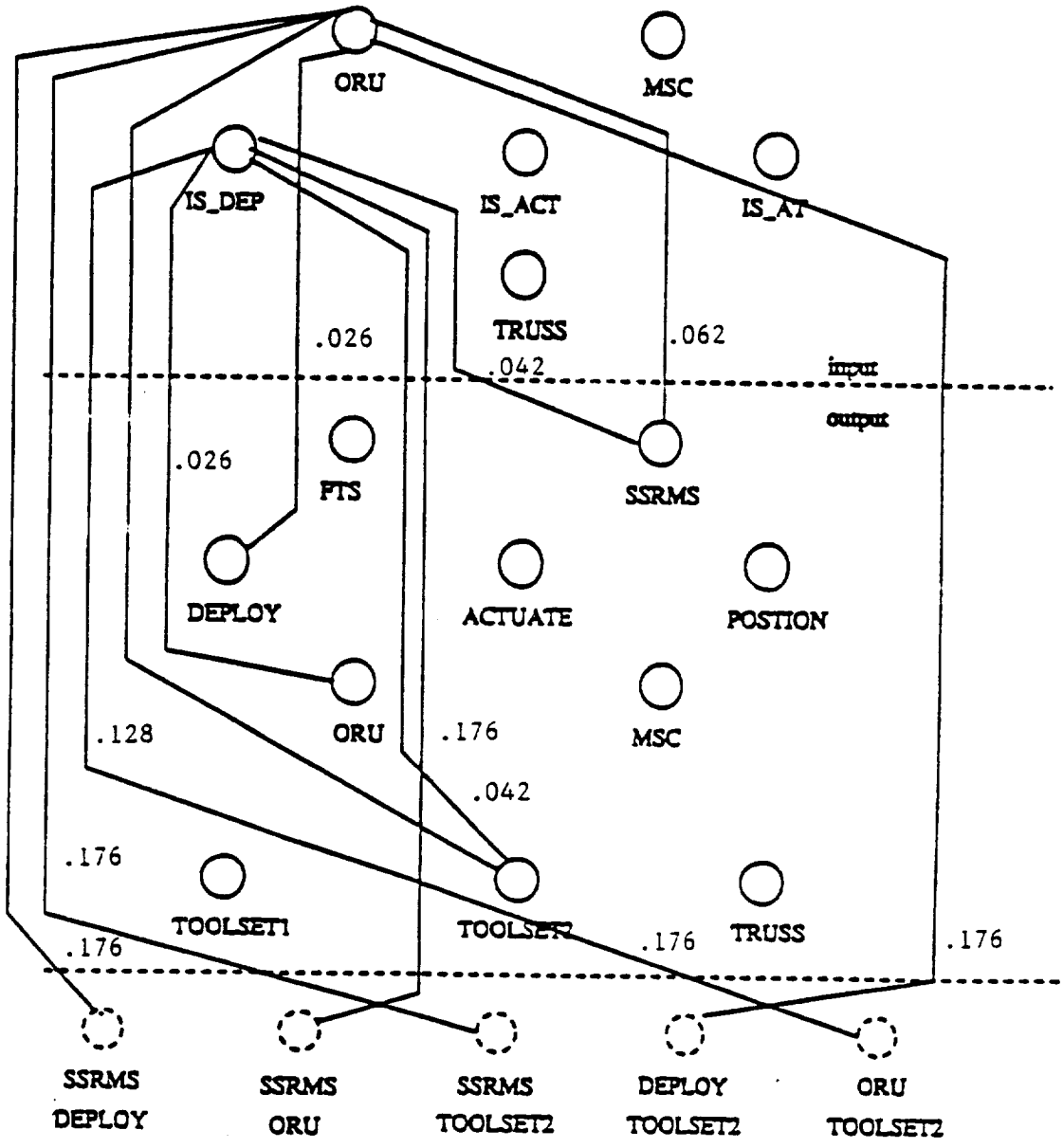


Figure 3.8: A training example with higher order nodes

FTS DEPLOY ORU TOOLSET1 -> ORU IS_DEPLOYED POE: 0.95
SSRMS DEPLOY ORU TOOLSET2 -> ORU IS_DEPLOYED POE: 0.30
SSRMS DEPLOY MSC TOOLSET2 -> MSC IS_DEPLOYED POE: 0.92
FTS ACTUATE ORU TOOLSET2 -> ORU IS_ACTUATED POE: 0.88
SSRMS POSITION ORU TRUSS -> ORU IS_AT TRUSS POE: 0.94
SPDM DEPLOY ORU TOOLSET2 -> ORU IS_DEPLOYED POE: 0.89

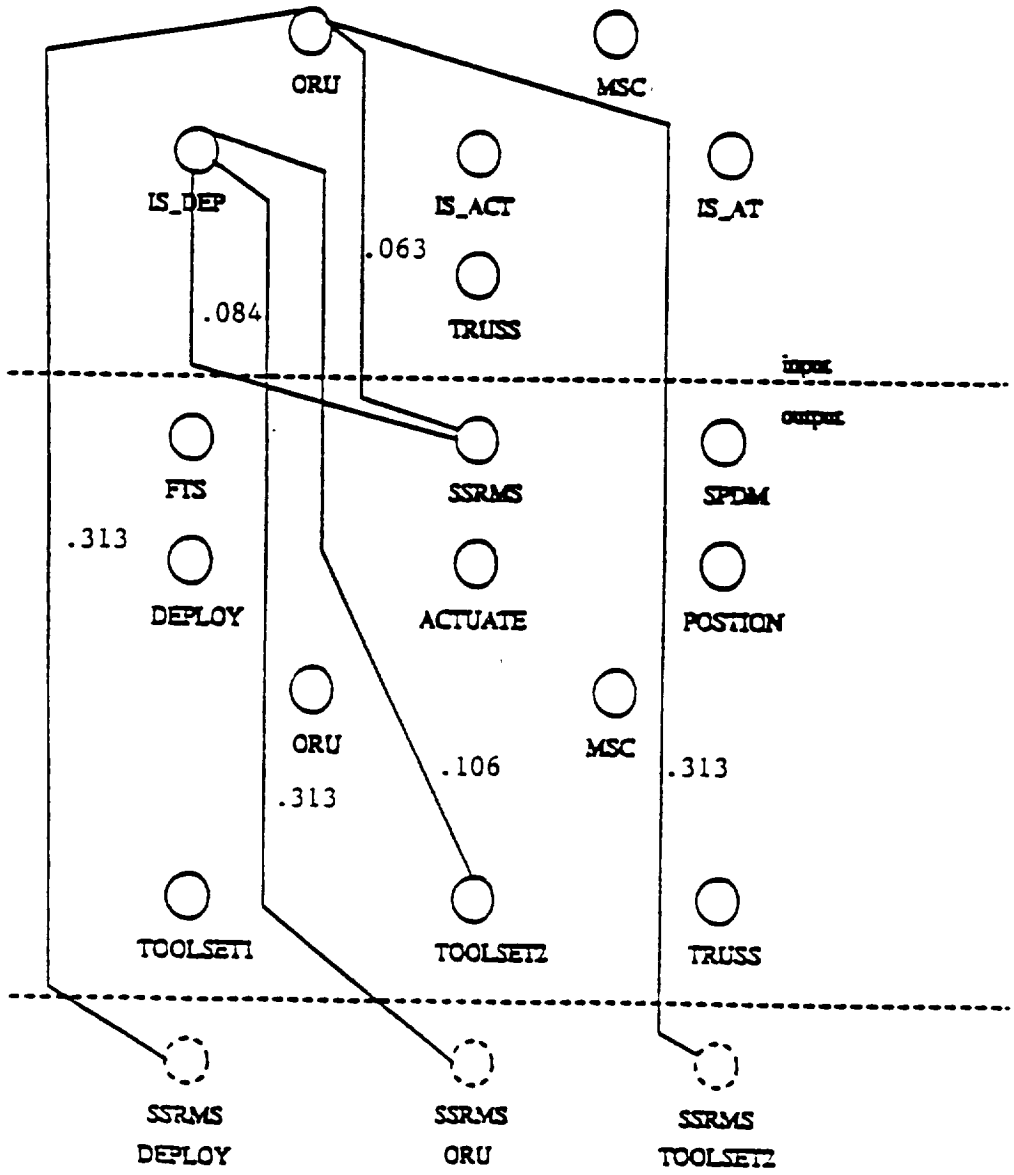


Figure 3.9: A second training example with higher order nodes

Table 3.1: Convergence and final error for test examples

Example	*Iterations*	*Max* $(G_\gamma)^2$	$\sum_\gamma (G_\gamma)^2$
3.6	342	0.1×10^{-9}	0.2×10^{-9}
3.7	781	0.3×10^{-9}	0.6×10^{-9}
3.8	457	1.8×10^{-9}	3.8×10^{-9}
3.9	536	4.3×10^{-9}	9.4×10^{-9}

- $0 \leq w_{ij} \leq 1$, the bounds on the weights.
- $\epsilon = 0.05$, the gradient step size.
- $Differror = 0.25$, the error band for rule selection.
- $\Theta_1 = 10^{-9}$, the minimum gradient before stopping.
- $\Theta_2 = 5 \times 10^{-3}$, the maximum allowable error for a rule in the training set.
- $\Theta_5 = 1 \times 10^{-4}$, the minimum weight before pruning is allowed.

The assignment of α_1 and α_2, which are the forcing function constants, is discussed below.

The number of gradient calculations required for each example, the square of the maximum error of a training set rule ($(G_\gamma)^2$), and the total sum square error for the training set ($\sum_\gamma (G_\gamma)^2$) are provided in the Table 3.1. The training technique led to extremely accurate representations for each of the rules, and converged quickly to the final weight values. These examples serve to illustrate the training procedure. Although the training sets presented here were small, more complicated ones have been tested. Chapter 6 provides a case study that includes a much larger and more difficult training set for the network.

3.4.1 Selection of training constants

As mentioned in section 3.3.1 of this chapter, the training algorithm requires the selection of constants α_1 and α_2 for the forcing functions. We know that $\alpha_1 < \alpha_2$ to insure that as much weight as possible is assigned to first order relationships. This section details the selection of these constants.

We can picture the function G as a quadratic. The constant α_1 determines the shape of the area around the minimum of the quadratic. If α_1 is too small, the area around the minimum is rather flat. This leads to a gradient that changes slowly as the descent procedure approaches the minimum of G. A slower gradient descent requires more descent steps. Therefore, if α_1 is too small, the training algorithm will take a long time to converge.

On the other hand, if α_1 is too large, the function G will sacrifice accurate representations of POE values in order to minimize the connection weights. This is unacceptable.

We can determine the maximum value of the forcing function constants for a given degree of accuracy. Let us define

- ϵ, a given weight step size.
- $A = \|E(\mathrm{N}_\gamma) - E_t(\mathrm{N}_\gamma)\|$, the desired accuracy.
- w_{max}, the maximum allowable weight value for any connection.
- α, a forcing function constant.

We want to insure that as a weight approaches its maximum value, the function G will decrease if the desired accuracy has not been met. If α is too large, G may increase due to the forcing function term, and the desired accuracy will never be reached. This becomes significant as a weight approaches its maximum value, where

the forcing function becomes the largest. In other words;

$$\{\|E(\mathrm{N}_\gamma) - E_t(\mathrm{N}_\gamma)\|\}^2 + \alpha(w_{max} - \epsilon)^2 > \{\|E(\mathrm{N}_\gamma) - E_t(\mathrm{N}_\gamma)\| - \epsilon\}^2 + \alpha w_{max}^2 \quad (3.31)$$

This is equivalent to

$$A^2 + \alpha(w_{max} - \epsilon)^2 > (A - \epsilon)^2 + \alpha w_{max}^2 \quad (3.32)$$

or

$$\alpha < \frac{2A - \epsilon}{2w_{max} - \epsilon} \quad (3.33)$$

If we let the step size approach 0, then

$$\lim_{\epsilon \to 0} \frac{2A - \epsilon}{2w_{max} - \epsilon} = \frac{A}{w_{max}} \quad (3.34)$$

So

$$\alpha < \frac{A}{w_{max}} \quad (3.35)$$

insures that we can represent a rule to a desired accuracy A given a bound on each weight, w_{max}.

Experiments were conducted to determine suitable values for α_1 across a range of training sets that required only first order representations. It was decided that magnitude of the error between the energy value of each asserted specific rule and the desired energy value of the rule should not be greater than 5×10^{-3}. This leads to an extremely accurate POE representation (within 99.5 percent of the desired value, as given by (3.8)) for the rules in the training set. Given this constraint, α_1 is bounded by 5×10^{-3} for $w_{max} = 1$, by (3.35). Experimentation showed that α_1 could assume any value in the range

$$5 \times 10^{-5} \leq \alpha_1 \leq 5 \times 10^{-3} \quad (3.36)$$

without requiring excessive gradient iterations (i.e., more than 10 times the number of iterations presented in Table 3.1).

We know that $\alpha_2 > \alpha_1$. Several experiments were conducted that varied α_2 while α_1 was fixed at 1×10^{-4} to examine the ability of the training procedure to eliminate second order nodes by allowing their weights to approach 0. Again, α_2 must be less than 5×10^{-3}. It was found that values of

$$1 \times 10^{-3} \leq \alpha_2 \leq 5 \times 10^{-3} \tag{3.37}$$

effectively eliminated non-essential second order nodes without sacrificing the POE representation in the network.

Based on this data, α_1 was set at 1×10^{-4} and α_2 was set at 4×10^{-3} for all training sets. The experimentation showed that α_2 had to be around 40 times larger than α_1 to develop an acceptable representation.

If third order nodes are required by the network, it is necessary to add a third forcing function with constant α_3. It is likely that the relationship between α_3 and α_2 would be similar in magnitude to the relationship present between α_2 and α_1 to develop an acceptable POE representation in the network. To accomplish this, α_1 would have to be reduced, along with α_2 to guarantee that $\alpha_3 < 5 \times 10^{-3}$. Although no experimentation was performed with third order nodes, one can see that this reduction in α_1 would significantly increase the number of gradient iterations required until convergence. If nodes higher than third order are necessary, the training time would increase even further. This demonstrates a weakness of this training technique.

3.5 Predicting POE Values for Untested Specific Rules

Prediction using the ARM is the ability to employ developed relationships between symbols to assign POE values to specific rules that have not been tested. When planning, the ability to predict POE values is essential since the number of tested specific rules is quite small compared to the number of possible specific rules. For a given planning situation, it is conceivable that no tested specific rule

provides an acceptable course of action. Instead, an untested rule must be selected to achieve a desired effect. To pick a "good" untested specific rule, therefore, one must understand the relationships between the symbols in that rule, and how these relationships affect its POE value.

As stated earlier, one of the difficulties in developing a training procedure for the ARM network is the fact that different specific rules share the same sets of symbols. Since this occurs, these rules also share the same sets of connections. While this makes training difficult, the sharing of connections forms the foundation for prediction.

The training procedure develops relationships between symbols by assigning weights to connections in the network. When untested specific rules overlap symbols with rules that are in the training set, their symbol relationships overlap as well. The weighted connections that form these relationships alter the energy value for the untested specific rule, which in turn alters the POE value for it. Therefore, the training procedure creates relationships that can be used to predict the POE value for untested specific rules.

Consider Figure 3.6. Any untested specific rule that contains $SSRMS$ or $TOOLSET2$ in the robotic action and ORU in the effect will have a decreased POE. Similarly, in Figure 3.7, the decreased POE is largely limited to untested rules that have $SSRMS$ in the robotic action and ORU in the effect. Relationships that alter POE values are also demonstrated in Figures 3.8 and 3.9.

Detailed examples of prediction are shown in the case study, which is presented in Chapter 6.

3.5.1 Untrained weights vs. zero weights

The training procedure creates large weights in some connections, and assigns small, or even zero weights to other connections. Some connections will approach

a weight of zero to insure that it accurately models a relationship between symbols. This indicates that the symbols work well together, and possess little or no inhibition.

On the other hand, the nature of the forcing function in G insures that if a connection is not used in the training set, its weight will also go to 0. This may be a problem, since it forces the network to assume that symbols have no inhibition if they are not part of the training set. This allows untested specific rules that have no connection overlap with the training set to assume a POE value of 1.0, or absolute certainty.

It is more appropriate that connections not used by the training set be assigned a default uncertainty value. After training, the connections that were not present in the training set could be assigned a nominal weight value, dictating the base uncertainty of an untested specific rule. To accomplish this, we must assign a base POE value to untested specific rules. Then, we can determine what each weight should be in the rule, by dividing the corresponding energy equally among all the first order connections for the rule. If a weight is left untrained by the training set, it would be assigned this weight value.

For example, if we assume a base POE value of 0.80 for untested specific rules, this corresponds to an energy of 0.223. If each rule contained 12 connections (e.g., 4 robotic action nodes connecting to 3 effect nodes), each untrained connection would be assigned a weight of $\frac{0.223}{12}$ or 0.018. All trained connections would maintain their trained weight values.

3.5.2 Examples of prediction in the ARM

This section uses the example presented in Figure 3.6 to demonstrate the prediction capabilities of the ARM. The example was extended to include other *ACTOR*s, *ACTION*s, and *OBJ*s. All of the connections not mentioned in the

training set for example 3.6 are assumed to be untested. A base POE value of 0.80 was assigned to completely untested rules, so the untested connections have a value of 0.018.

The general rules used are as follows.

- $< manipulator >\ ACTUATE\ < object >\ < tool > \rightarrow$
  $< object >\ IS-ACTUATED\ NULL$
- $< manipulator >\ ATTACH\ < object1 >\ < object2 > \rightarrow$
  $< object1 >\ IS-ATTACHED-TO\ < object2 >$
- $< manipulator >\ DEPLOY\ < object >\ < tool > \rightarrow$
  $< object >\ IS-DEPLOYED\ NULL$
- $< manipulator >\ POSITION\ < object >\ < location > \rightarrow$
  $< object1 >\ IS-AT\ < location >$
- $< transporter >\ TRANSPORT\ < object >\ < location > \rightarrow$
  $< object1 >\ IS-AT\ < location >$

The value of a general rule inhibitory connection was 1.0. Each of the following specific rules was asserted on the network, and the network responded with the POE values shown.

1. $FTS\ DEPLOY\ ORU\ TOOLSET1 \rightarrow$
   $ORU\ IS-DEPLOYED\ POE:\ 0.950$
2. $SSRMS\ DEPLOY\ ORU\ TOOLSET2 \rightarrow$
   $ORU\ IS-DEPLOYED\ POE:\ 0.300$
3. $SSRMS\ DEPLOY\ MSC\ TOOLSET2 \rightarrow$
   $MSC\ IS-DEPLOYED\ POE:\ 0.920$

4. *MT TRANSPORT AWP TRUSS* →
   AWP IS – AT TRUSS POE: 0.800

5. *SPDM TRANSPORT AWP TRUSS* →
   AWP IS – AT TRUSS POE: 0.295

6. *SSRMS POSITION ORU TRUSS* →
   ORU IS – AT TRUSS POE: 0.487

7. *SPDM POSITION ORU TRUSS* →
   ORU IS – AT TRUSS POE: 0.815

8. *FTS POSITION ORU TRUSS* →
   ORU IS – AT TRUSS POE: 0.830

9. *JRMS DEPLOY AWP TOOLSET2* →
   AWP IS – DEPLOYED POE: 0.858

10. *JRMS DEPLOY ORU TOOLSET2* →
    ORU IS – DEPLOYED POE: 0.514

11. *SPDM DEPLOY MSC TOOLSET1* →
    MSC IS – DEPLOYED POE: 0.946

The above responses have the following explanations.

1. This specific rule is in the training set.

2. This specific rule is in the training set.

3. This specific rule is in the training set.

4. No connections used by this specific rule are in the training set. Therefore, each weight in the rule was assigned a default value producing the default POE value of 0.80.

5. The $SPDM$ is not a member of the class $<$ *transporter* $>$ so this specific rule violates a general rule. The violation adds 1.0 to the energy value of the configuration, and results in a low POE value.

6. This rule demonstrates how the inhibitory connection between $SSRMS$ in the robotic action and ORU in the effect lower POE values for untested rules that use this connection.

7. The ORU robotic action to ORU effect connection in this rule has assumed the value 0 after training. All other connections are untrained, and possess the base weight value. Therefore this specific rule has a POE slightly above the base untrained POE value.

8. Two connections in this specific rule have been set to 0 by the training set.

9. This specific rules demonstrates the use of several symbols that work well together along with some untested connections.

10. This rule violates the $STATE$ to $ACTION$ relationship given by the general rules.

11. This specific rule employs many trained connections that work well together, as specified by the training set.

Overall, we see that the ARM can provide consistent predictive POE values for untested specific rules.

3.6 Extensions to the ARM model

Two features are added to the ARM model to make it more versatile. The first feature allows the user to encode knowledge into the ARM network, if so desired. The second feature provides a measure of confidence to the user for the POE value of a specific rule.

3.6.1 The Knowledge Set

From previous discussion, it is known that the ARM is trained on specific rules that have been tested in the robotic environment. The training set is constructed by the user to evaluate the ability of agents to work together and achieve a change in state of an object in the world. It is likely, however, that particular combinations of agents will not be tested if the user knows that they function poorly together, or lead to "disaster" situations. Therefore, the training set may not include information about symbols combinations that are known by the user to be avoided. If these combinations are not trained on the network, their connections are assigned nominal weights by the base POE assignment procedure outlined in section 3.5.1. Unfortunately, this may allow the ARM to assign a high POE value to untested specific rules that contain subsets of these avoided symbol combinations.

To remedy this problem, a feature is added to the ARM network that allows the user to encode information about symbol combinations that should be avoided. This feature is called the *knowledge set*, and is composed of a set of *knowledge rules* in which the user encodes the information. A knowledge rule is created from a general rule by providing set of symbols or symbol classes in the robotic action that should not be used together to achieve a set of symbols or symbol classes in the effect. For example, given the general rule

$$< dex > \ ACTUATE \ < obj > < tool > \rightarrow$$

$$< obj > \ IS-ACTIVATED \ NULL$$

the knowledge rule

$$< dex > \ NULL \ NULL \ TOOLSET1 \ \rightarrow$$

$$NULL \ IS-ACTIVATED \ NULL \qquad (3.38)$$

is employed to show that dextrous manipulators should not be used with *TOOLSET1* when trying to achieve the state $IS-ACTIVATED$. Similarly, given the general rule

$$< dex > \quad ATTACH \quad < obj > \; < obj > \; \rightarrow$$

$$< obj > \quad IS-ATTACHED-TO \quad < obj >$$

the knowledge rule

$$FTS \; ATTACH \; NULL \; NULL \; \rightarrow$$

$$NULL \; NULL \; TRUSS \tag{3.39}$$

states that employing the *FTS* to *ATTACH* should not be attempted when the *TRUSS* is the indirect object of the effect. The *NULL* symbols in each of the above knowledge rules are simply placeholders.

The knowledge rules are mapped onto the ARM network using a technique similar to general rule mapping. For a first order relationship (only one non-NULL level in the robotic action of a knowledge rule), a high weight inhibitory link is created from each symbol in the specified symbol class of the robotic action to each symbol in the symbol class designated in the effect of the knowledge rule. For second order relationships, (two non-NULL levels specified in the robotic action), second order nodes are created. These nodes are formed by a pairwise combination of each symbol in the first specified symbol class of the robotic action of the knowledge rule, with each symbol in the second specified symbol class. A high weight inhibitory connection is created from each of these second order nodes to each symbol in the symbol class designated in the effect of the knowledge rule. For higher order nodes, the process is the similar.

The knowledge set, therefore, provides the user with another mechanism for specifying relationships between symbols in the network.

3.6.2 The Confidence Factor

When a specific rule is asserted on the ARM network, the network responds with the POE value of the asserted rule. If the specific rule is not a member of the training set, the POE value is predicted from

- weights developed by rules in the training set that overlap the asserted specific rule, and
- weights that are not developed by the training set, but are assigned base probability values, as described in section 3.5.1.

If an asserted untested specific rule contains connections that are not in the training set, some of its POE value comes from the base probability weights. Since these connections are not influenced by the training set, however, it is unknown what their actual weights should be. Although the base probability weight is used to approximate expected behavior, it may not provide adequate representation when an untested specific rule contains many untrained connections, and has to rely largely on the base probability weights. Unfortunately, the POE value alone does not allow a user to discern untested specific rules which contain a large number of untrained connections.

If an untested specific rule shares many of the same symbols with a specific rule in the training set, it is very similar to the tested rule. The more similar an untested rule is to a tested one, the more likely the POE value of the untested rule can be reliably predicted. On the other hand, the less similar an untested rule is to any rule in the training set, the less reliable its predicted POE value may be. Unfortunately, the POE value of an untested specific rule does not provide a measure of the overlap between the rule and each specific rule in the training set.

To provide the user with a "measure of confidence" in the POE value for an untested specific rule, the *confidence factor* (cf) is presented. In this implementation, the cf of an untested specific rule is a function of:

1. The percentage of trained connections in the untested specific rule.
2. The maximum of the percentages of the connection overlaps with each specific rule in the training set.

It is possible to define other valid confidence measures, if the user desires.

Mathematically, the cf used in this case study is given by:

$$cf(\mathbf{N}) = \sqrt{\frac{C_t(\mathbf{N})}{C} \cdot \frac{C_o(\mathbf{N})}{C}} \tag{3.40}$$

where

- N is a particular specific rule asserted on the network.
- C is the total number of first order connections for the specific rule in the ARM network.
- $C_t(\mathbf{N})$ is the number of trained first order connections for specific rule N
- $C_o(\mathbf{N})$ is found be determining the number of connections which are the same between N and each rule in the training set, and then selecting the maximum of these values.

Using this scheme, a rule in the training set has a cf of 1.0, which is the maximum cf value, and represents total confidence in the POE value. All untested specific rules have values between 0 and 1. The larger the cf for an untested specific rule, the more confident the user can be that the POE value accurately reflects the symbolic relationships in the rule.

3.7 Conclusions

This chapter detailed the design of the Associative Rule Memory, as presented in Figure 3.10. The main contributions of the ARM are as follows.

1. The design of a neural network model that is able to represent a symbolic grammar comprised of a robotic action and effect.

2. The ability of this model to maintain instantiations of the grammar with a real valued number representing the probability that the robotic action achieves the desired effect.

3. A training procedure that guarantees that the network will develop accurate POE representations for all specific rules in the training set.

4. A training procedure that develops weighted connections that represent the reliability with which a robotic action symbol affects an effect symbol.

5. A technique for adding higher order nodes when necessary, and pruning them when they are unnecessary.

6. A demonstration that the training procedure builds connections that can be used for predicting POE values for untested specific rules.

As stated in the chapter, the ARM model is bound by the following constraints.

1. The POE value is a function of the agents in the robotic action and effect, and has not been altered by any other environmental influences.

2. A one-to-one symbol to node mapping is used instead of a distributed representation. This leads to a large number of nodes, but simplifies the symbol relationships stored in the connections of the ARM.

3. The development of hidden nodes may lead to slower training of the network if third, fourth or even higher order nodes are required for accurate POE representation.

Overall, the training examples demonstrated that the network could develop extremely accurate POE representations for the rules in the training set. Also, the training examples allowed us to explore the predictive capabilities of the ARM, which allows the exploitation of implicit relationships in the training sets, and produces reasonable POE values for untested specific rules.

This chapter also presented the knowledge set and the confidence factor, two enhancements to the ARM model. The knowledge set allows the user to encode known relationships in the world that are not present in the training set. The cf provides the user with a measure of confidence in the POE value of an untested specific rule.

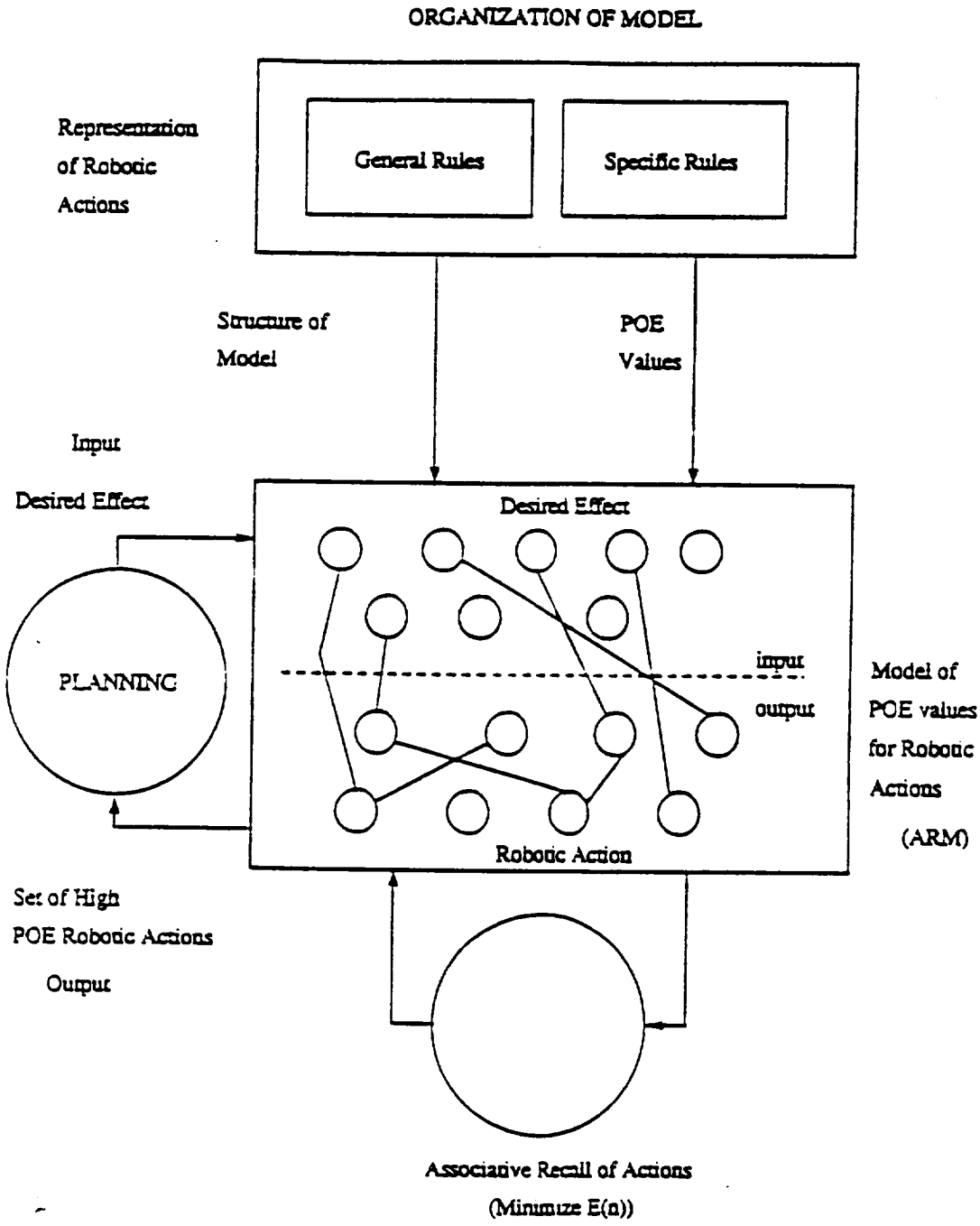


Figure 3.10: Block diagram with ARM displayed

CHAPTER 4
ASSOCIATIVE RECALL - AN OPTIMIZATION TECHNIQUE

The functions of the Associative Rule Memory are as follows.

1. Storage of specific rules.
2. Storage of general rules.
3. Generalization of tested specific rules to predict POE values for untested specific rules.
4. Associative recall of high POE robotic actions given a desired effect.

This chapter examines the last of these ARM functions, associative recall as shown in the block diagram 2.1.

Associative recall is the process of extracting a stored inference or trace from a memory by providing the memory with a "key" that has been matched to the inference. For this research, the memory key is a desired effect that must be achieved by the robotic system. The stored inference in the ARM is a robotic task that achieves the desired effect. In other words, by providing the ARM with a desired effect as input, the ARM should produce as output a robotic task that has a high probability of achieving the desired effect, along with the probability of effect.

In artificial neural networks, associative recall is performed by asserting a set of input nodes, the key, and allowing network to "settle" the output nodes on a state that is the matched memory trace. Most ANN's find the associated memory trace by optimizing an energy measure that is a function of the weights in the network and the current state of the nodes. Examples of networks that perform recall in this manner are Hopfield Networks [44] and Boltzmann Machines [55].

Both of these ANN's represent consistency between asserted nodes by the energy formula given in (3.3). For these networks, a high energy value implies a large inhibition between sets of asserted nodes. Large inhibition is present when the memory trace asserted on the output nodes does not match the key presented on the input nodes.

The lower the value of $E(\mathrm{N})$ for a particular input key, the smaller the inhibition that exists between the key and the recalled inference present on the output nodes of the network. Therefore, associative recall is the process of finding the set of asserted output nodes that minimizes the function $E(\mathrm{N})$ for a given set of asserted input nodes.

In the previous chapter, we developed a method for storing the probability of effect for provided specific rules in the ARM, and demonstrated the generalization of these probabilities to rules that have not been explicitly tested. The procedure uses the weights of the network to store the probability of a specific rule as an energy value. Since the relationship between probability of effect and energy is given by

$$POE(\mathrm{N}) = e^{-E(\mathrm{N})} \tag{4.1}$$

we can see that a high probability implies a low energy value. Specifically, a probability of effect of 1.0 has an energy value of 0.0, and the energy increases for all probabilities less than 1.0. Further, violations of general rules incur an increased energy because each violation adds positive weight to the value of $E(\mathrm{N})$.

The design of the ARM implicitly allows associative recall to be performed in ways similar to the ANN's discussed above. By minimizing the value $E(\mathrm{N})$ of the ARM for a particular desired effect (input nodes), the network settles on a set of robotic action (output) nodes that have a high probability of achieving the desired effect. The procedure is described as follows.

1. Assert desired effect nodes (Input).

2. Choose a set of robotic action nodes (Output).

3. Assert robotic action nodes and higher order nodes.

4. Calculate $E(\mathbf{N})$. Compute $POE(\mathbf{N})$.

5. If $POE(\mathbf{N})$ is less than desired (non-optimal) go to 2.

6. End. Robotic action that has a high probability of achieving the desired effect has been found.

Step 2 of the above algorithm is accomplished through an iterative optimization technique. In general, given a function $f(\cdot)$ defined on a n-dimensional space Y^n, an optimization technique attempts to find suitable values for the vector $\mathbf{X} = (x_1, x_2, \cdots, x_n) \epsilon Y^n$ such that $f(\mathbf{X})$ achieves a desired value, often the minimum or maximum of the function. An iterative optimization function considers the past history of attempts at optimizing $f(\cdot)$ when choosing the next search point $\mathbf{X}$.

The rest of this chapter is devoted to the optimization technique required by step 2 of the associative recall algorithm. Section 4.1 describes the shape of the energy hypersurface, and the requirements it places on an optimization technique. Section 4.2 compares two suitable optimization techniques, the Genetic Algorithm and Simulated Annealing. Section 4.3 describes research on reducing the search time of the Genetic Algorithm using an immigration operator. Section 4.4 develops a proof that shows that the GA modified with the immigration operator converges in probability to the optimum of a cost function. Section 4.5 explores representation issues between the GA and the ARM model. Section 4.6 describes a method for find sets of high POE robotic actions for a given a desired effect. Section 4.7 outlines the contributions and concludes this chapter.

4.1 The ARM Energy Function

The ARM energy function is defined by

$$E(\mathbf{N}) = \frac{1}{2}\sum_i \sum_j w_{ij} n_i n_j \tag{4.2}$$

where n_i is the state of node i (0 or 1) and w_{ij} is the weight of the connection between nodes i and j. This function maps a binary vector $\mathbf{N} = (n_1, n_2, \cdots, n_k)$, $(n_x \in \{0,1\})$ onto a real number bounded below by 0.0.

Several considerations are essential when optimizing this function.

1. The domain of the energy function is discrete and binary.

2. The range of the energy function can have severe discontinuities for neighboring domain values. Neighboring domain values have a Hamming Distance of 1.

These two constraints rule out most numerical optimization techniques, such as discrete gradient descent, that would quickly terminate in a locally optimal energy state. Instead, probabilistic search methods are used to find globally optimal energy values for the ARM. These methods provide means of escaping or avoiding the difficult local minima present in the ARM, and allow the function domain to be binary.

4.2 Two Optimization Techniques

Two algorithms that have often been used to solve difficult optimization problems are Simulated Annealing (SA) and the Genetic Algorithm (GA). Both use a probabilistic selection and generation strategy to develop search points for evaluation.

4.2.1 Simulated Annealing

Simulated Annealing is the most common technique used to minimize the energy function in a Boltzmann machine during the process of associative recall. This technique simulates the annealing process of metal by probabilistically allowing uphill steps in a state–dependent cost function while finding the global cost minimum, or ground state. The algorithm allows control of the search randomness by a user specified parameter, T. In true metal annealing, this cost function is the energy of the system, E, and T is the annealing temperature [57].

Given a small random change at iteration i from the system state $\mathrm{N_i} = (n_1, n_2, \ldots, n_k)$ to $\mathrm{N_i'}$ and the resulting energy change, $\Delta E = E(\mathrm{N_i'}) - E(\mathrm{N_i})$, if $\Delta E \leq 0$, the change is accepted. If the change is accepted, the current state $\mathrm{N_{i+1}}$ is set equal to the new state $\mathrm{N_i'}$. If $\Delta E > 0$, the probability the new state is accepted is

$$P(\mathrm{N_{i+1}} = \mathrm{N_i'}) = e^{-\Delta E/K_B T} \tag{4.3}$$

where K_B is the Boltzmann Constant and T is a user set parameter. By reducing T along a schedule, called the annealing schedule, the system should settle into a near–ground state as T approaches 0.

Another method for SA is discussed in [55]. Using this method, if the energy change between $\mathrm{N_i}$ and $\mathrm{N_i'}$ is ΔE, then regardless of the previous state, accept state $\mathrm{N_i'}$ with probability

$$P(\mathrm{N_{i+1}} = \mathrm{N_i'}) = \frac{1}{1 + e^{-\Delta E/T}} \tag{4.4}$$

Since the domain of the ARM is binary, it should be noted that in both of the above methods, $\mathrm{N_i'}$ is Hamming distance 1 from $\mathrm{N_i}$.

The process of Simulated Annealing escapes local minima through its probabilistic search, and converges to the global energy minimum in probability under conditions detailed in [69]. These conditions force the annealing schedule to follow a

exponential decay temperature trajectory, so the convergence to the global minimum can be extremely slow. Further, SA is an uninformed optimization technique that cannot exploit the implicit constraints that may be present in the target function.

4.2.2 The Genetic Algorithm

Another technique used to optimize nonlinear or discontinuous functions is the Genetic Algorithm (GA) [70]. In contrast to other random search techniques, the GA maintains a population of points in the space while searching for the optimum. For most GA's, each point in the domain is represented by a binary string and has an associated fitness value obtained by evaluating the cost function at that point. Since the makeup of the population is changed each iteration to emphasize members (points) that optimize the cost function, a near-uniform population will develop as the GA searches for the optimum string.

Each cycle of the GA is comprised of four main phases: evaluation, selection, recombination and replacement. During evaluation, each member is assigned a fitness value relative to its cost, such that lower cost members receive higher fitness values. Based on fitness, members are probabilistically selected from the population for recombination. These members are called parents. Parental pairs exchange bits during the recombination process and form binary strings called children. This process is called *crossover*. The worst members of the population are replaced by the children, and the cycle repeats. Details of the algorithm are presented in [71].

In an attempt to prevent population convergence to local optima (premature convergence), a *mutation* operator is added to the system. With a new generation of the population, each bit of every member has a small probability of inverting. The mutation adds diversity to the population and promotes local search and hill-climbing.

Particular aspects of this algorithm make it a powerful search tool. The

crossover mechanism forces search on an n-dimensional hypercube by discovering and promoting particular substrings (called *schemata*) that perform well. The schemata are low-order substrings, where the *order* or a substring is its length in bits. These schemata combine to discover the structure of the search space, which may not be known initially. The discovery and propagation of high performing schemata allows the GA to exploit implicit constraints in the target function. Further, since the algorithm uses a population of points, many planes of the hypercube can be searched at once, leading to implicit parallelism [70]. Applications of this algorithm to optimization problems have been presented in [72, 87, 88].

4.2.3 Some initial experiments: comparing SA and GA optimization techniques

Some initial experiments were performed to show that the Genetic Algorithm could be used as an optimization technique for Boltzmann Machines, including the ARM. These results were originally described in [16].

A Boltzmann Machine is created containing 15 nodes, $\mathbf{N} = (n_1, n_2, \cdots, n_{15})$. Each node is connected to every other node. Nodes n_4 and n_6 form the input to the network, and their values are fixed at 1.0. The other network nodes form the output of the network. By changing the values of the output nodes in the network, the minimum energy of the network can be found. For this purpose, SA and GA optimization techniques are invoked to find the minimum energy by altering the output node values.

For the given input, the net has three energy minima corresponding to states $\mathbf{N} = \{(001010100100100), (110110110001101), (001111101100010)\}$. The respective energy for each of these three states is (0.8, 0.6, 1.0). Each simulation technique attempts to find the global energy minimum of the net, which is 0.6, and corresponds to the correct output for the given input. The cases presented here show best and

worst performance of each technique over 10 trials.

The Genetic Algorithm is set at a population size of 20 members. Each member is 15 bits long, and represents a complete state of the network. Consequently, each bit of a member represents the state of one node in the network.

The fitness function assigns values by the relationship

$$FITNESS(s) = MAXCOST - COST(s)$$

where s is a member in the population, $MAXCOST$ is the maximum energy of the network, and $COST(s)$ is the cost of the network if it assumed the state given by member s. One-point crossover is used and the mutation rate is set at 0.005 mutations per bit. Each cycle selects a set of parents for crossover equal to 80 percent of the population size. Further, the GA is enhanced by replacing the worst member every two generations with a random member, a detail that will be discussed below.

Some initial experiments with Simulated Annealing using a heuristic cooling schedule yielded suboptimal solutions when minimizing the energy of the network. To prevent suboptimal solutions from occuring during these experiments, Simulated Annealing is performed using the acceptance criteria in (4.4). The system is cooled in accordance with the law

$$\frac{T_1(t)}{T_0} = \frac{1}{\log(10+t)} \tag{4.5}$$

where $T_1(t)$ = temperature at time t and T_0 = initial temperature.

The net state changes in Hamming distance 1 increments.

Figures 4.1, 4.2, 4.3 and 4.4 present the best and worst performance of each algorithm over 10 trials. The GA found the minimum energy string between the 20th and 180th population. Since there were 20 strings per population, this indicates that between 400 and 3600 search points had to be generated. The best performance by

Simulated Annealing required over 5500 generated points. The worst performance did not find the minimum after evaluating 12000 generated points, which was the most attempted.

The results of these limited experiments clearly demonstrate the ability of the Genetic Algorithm to perform the task of associative recall in a Boltzmann Machine at least as well as Simulated Annealing. Also, the GA is able to exploit the implicit constraints associated with a problem, such as the relationships between nodes in the ARM, which Simulated Annealing cannot do. Further, Simulated Annealing is controlled by an exponential decay temperature trajectory in order to guarantee convergence to the global optimum solution, which can inhibit SA from finding quick solutions to easy problems. For these reasons, the Genetic Algorithm seems to be a better optimization technique than Simulated Annealing for associative recall of the ARM. Therefore, we have chosen the GA as the optimization technique for the ARM. This is shown in Figure 4.5.

4.3 Reducing the Search Time of a Genetic Algorithm

The next several sections present and analyze a new technique for reducing the number of function evaluations required by the GA to find the global optimum solution. Although the test suite used in experimentation contained a broad class of functions, direct application can be made to Boltzmann Machines and the ARM.

4.3.1 An introduction to immigration

The tradeoff between exploration and exploitation in serial Genetic Algorithms for function optimization is a fundamental issue [71]. If a GA is biased towards exploitation, highly fit members are repeatedly selected for recombination. Although this quickly promotes better members, the population can prematurely converge to a local optimum of the function. On the other hand, if a GA is biased towards

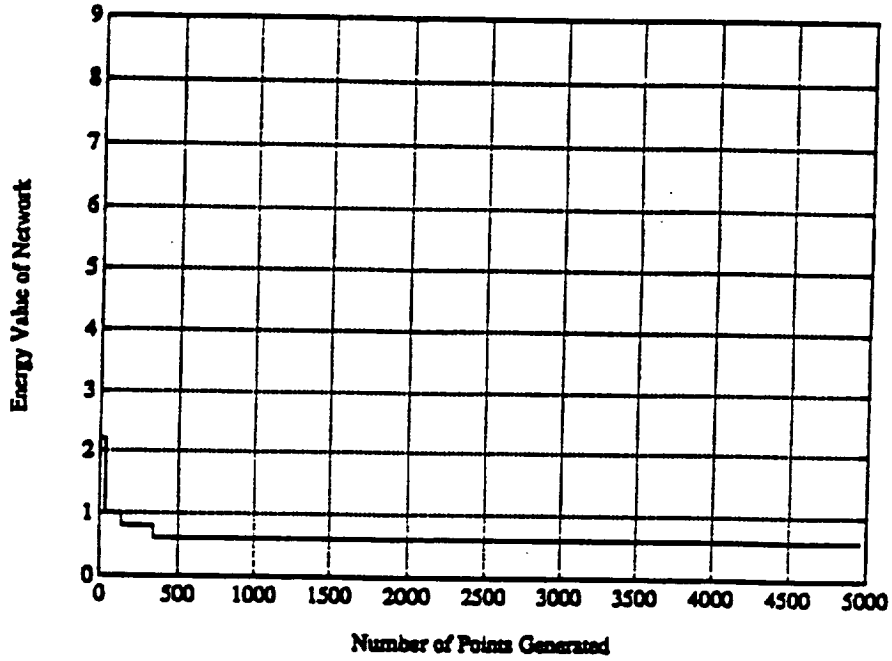


Figure 4.1: Best performance of GA

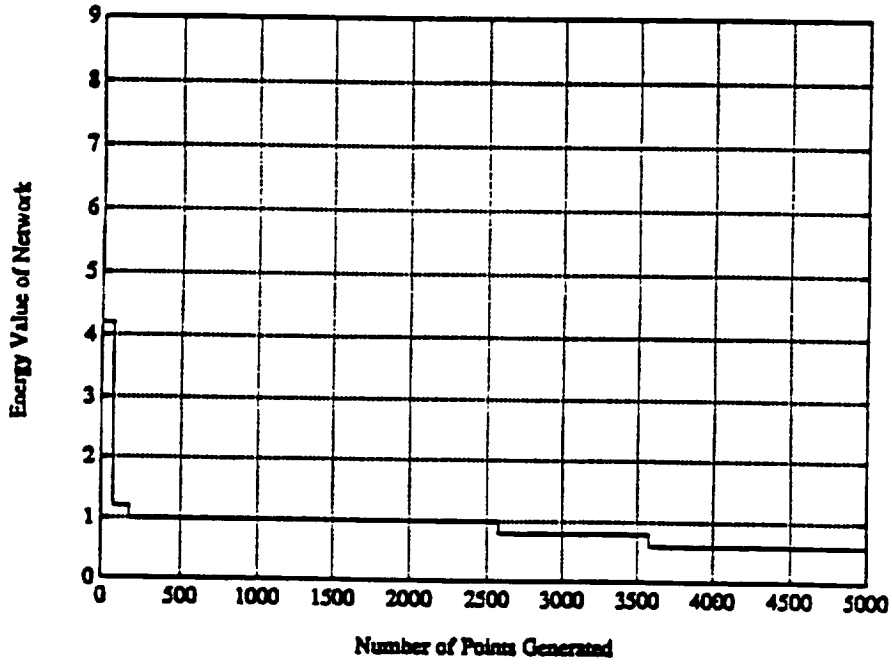


Figure 4.2: Worst performance of GA

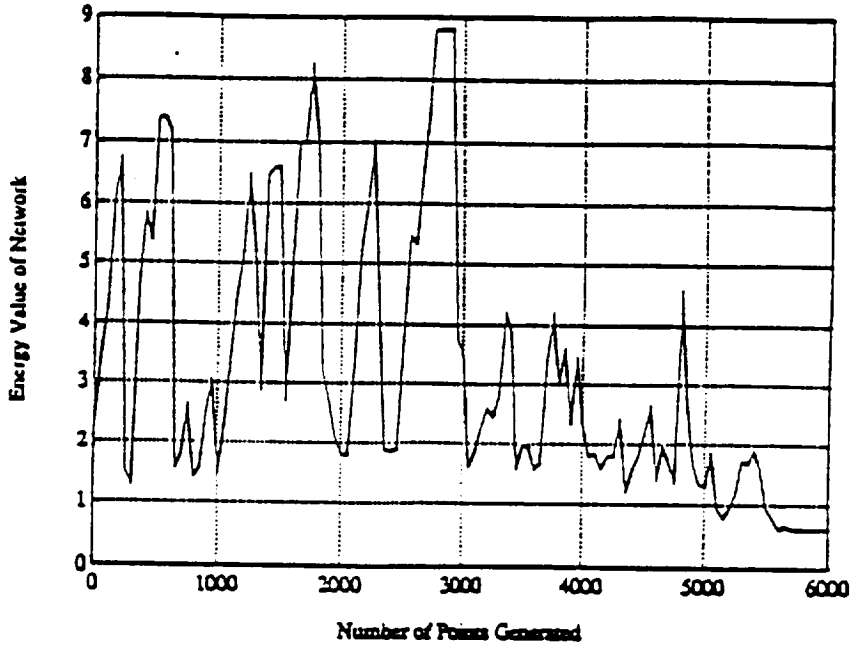


Figure 4.3: Best performance of SA

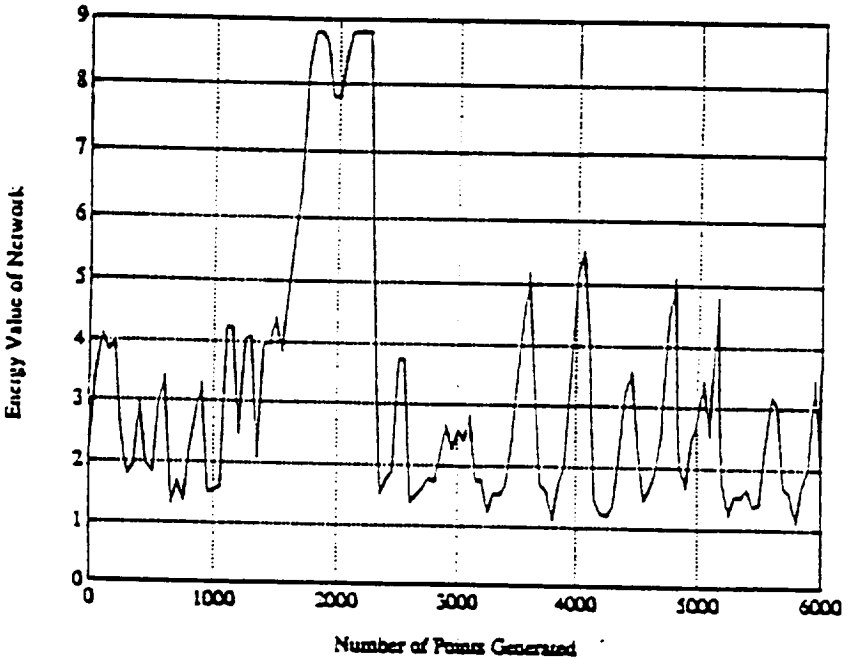


Figure 4.4: Worst performance of SA

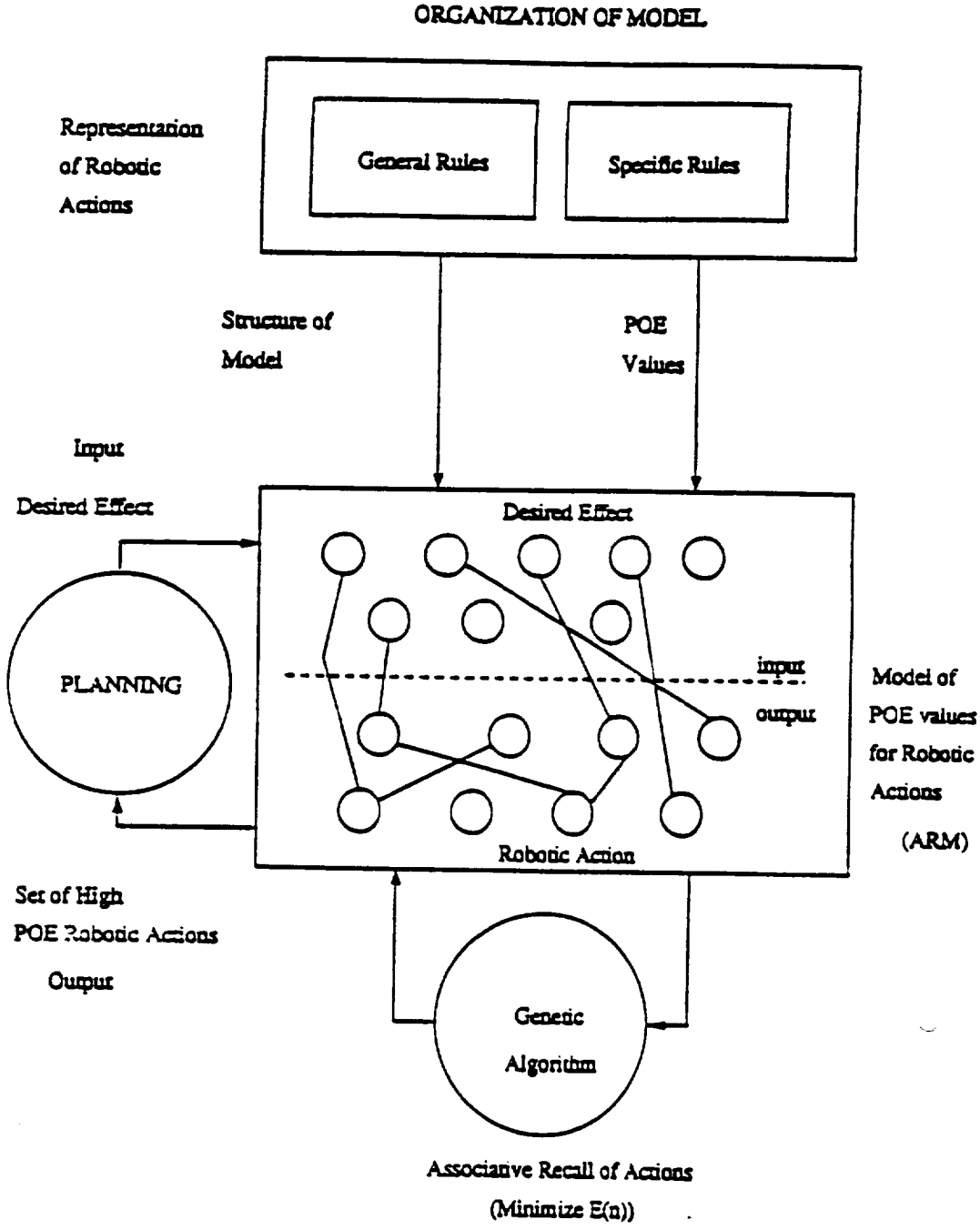


Figure 4.5: ARM system block diagram with GA for associative recall

exploration, large numbers of schemata are sampled which tends to inhibit premature convergence. Unfortunately, excessive exploration results in a large number of function evaluations, and defaults to random search in the worst case. To search effectively and efficiently, a GA must maintain a balance between these two opposing forces.

This study experimentally examines an *immigration* operator that for certain types of functions, allows increased exploration while maintaining nearly the same level of exploitation for the given population size. Section 4.3.2 provides relevant background material on this topic and develops the motivation for this study. In section 4.3.3, we describe the immigration operator, its incorporation into the evaluation, selection and recombination cycle of a Genetic Algorithm, and predict the behavior of this "Modified" Genetic algorithm. In section 4.3.4, the implementation of two genetic algorithms is described. One algorithm is based on steady state GA's, and the other is based on restarted GA's as described by Goldberg [78]. Also described is the implementation of the two GA's modified with the immigration operator.

To compare the performance of each GA with and without immigration, a suite of test functions is developed. Each function is characterized by different types of local and global optima. The local optima are designed to provide traps that the genetic algorithm must successfully avoid or recover from to achieve a global optimum. These functions are defined in section 4.3.5. Section 4.3.6 discusses the experiments performed on the test suite and presents each GA experiment in terms of fitness assignment, population sizes, mutation rate and immigration rate. The results of the GA experiments are examined in terms of the number of evaluations required to find the global optimum of each function and are presented in Section 4.3.7. It is shown that a GA modified with immigration reduces the average number of evaluations required to find the function optimum over a range of population

sizes. It is further shown that the number of trials requiring an excessive number of evaluations is reduced for functions in this set. Section 4.3.8 provides conclusions and recommendations for further research on this topic.

4.3.2 Background and motivation

Population size is one parameter that directly effects the balance between exploration and exploitation. DeJong [72] notes that increasing the population size improves long-term performance of a GA at the expense of degraded on-line performance for his test suite. In an extensive study, Schaffer et al. [89] test GA parameters for an expanded suite of functions and measure on-line performance. The study indicates that functions with many local optima have good on-line performance with larger population sizes; however, the study notes that an excessively large population imposes an increased number of evaluations per generation and produces poor overall performance.

As these studies show, increasing the population size of a GA supplements the amount of "raw material" available for processing. If the necessary material is present in the initial population, the GA can converge to an optimal solution. If the initial population size is very large, the optimal schemata may very likely be present and the optimal solution will be found.

However, large population size can lead to an inefficient GA. The GA processes the schemata contained in the "raw material" and exploits those schemata that perform well. As the population size grows and the selection "pie" is divided into more slices, the exploitation decreases. A decrease in exploitation means that better schemata propagate at a slower rate. This forces the GA to increase the number of population samples over repeated generations in order to determine the optimal schemata, and slows convergence of the GA.

Goldberg [78] also emphasizes the tradeoff between schema processing (exploration) and convergence rate (exploitation) with regard to population size. To combat slow convergence in serial GA's while finding the optimum of a function, Goldberg suggests using small population GA's that are restarted after convergence. The restart procedure consists of keeping only the best individuals of the converged population and replacing other members by randomly generated individuals. Goldberg shows that this procedure maintains a high rate of schema processing, a cost measure developed to examine GA performance. Intuitively, this technique maintains a very high level of exploitation, since small populations rapidly converge to the best schemata present. Exploration is also enhanced by restarting the population with random members after convergence. Therefore, this technique seems to achieve some balance between the two GA forces.

Given that small population GA's may not possess the necessary schemata to find the optimal solution, and large population GA's can be inefficient in schema processing, this study examines the effect of continually replacing the worst members of a GA population with random members. The technique is called *immigration*. For GA's of the type tested by DeJong [72] and Schaffer et. al. [89], immigration increases the amount of "raw material" available to the GA, and enhances exploration without increasing population size. For Goldberg's restarted GA, immigration allows increased exploration while the population converges, and may prevent quick convergence to a local optimum. This is especially significant if the test function contains local optima that are difficult to escape.

4.3.3 A GA with the Immigration Operator

To balance exploration with exploitation, we propose the following algorithm that incorporates the immigration operator into the general structure of a Genetic Algorithm. A preliminary version of this algorithm was originally presented in [16].

The Modified Genetic Algorithm

1. Evaluate each member of the population and assign a fitness value.
2. Replace m current worst members of the population with m randomly generated and evaluated members. (Immigration Operator)
3. Probabilistically select a subset of members based on fitness.
4. Recombine selected members to form children.
5. Replace the worst members of the population with children.
6. Mutate some members to maintain population diversity and perform local search. Mutation is not performed on one copy of the current best member.

With each generation of the GA, random members are immigrated and replace the worst members in the population. It is important to note that the number of random individuals substituted into the population each generation (m, called the *immigration rate*) is small compared to the size of the population. The advantages and tradeoffs of immigration are described below.

1. When a non-modified GA is initialized its population of n random members must contain most of the "raw material" required to assemble the optimal string through selection and crossover. With smaller population GA's, the necessary schemata to build the optimal string may not be present in the initial population. If this is the case, the GA must rely on mutation to bring in the necessary schemata, which can be very inefficient, especially in problems with many local optima. The immigration operator allows the GA to sample many more individuals during search, and more easily acquire the necessary structure to find the optimal string; however, immigration does not increase the size of the population, since random members replace poor performers.

Therefore, *immigration allows a size n population to explore the space of a larger population.*

2. Since the actual population size is not increased to accomplish the added exploration, the high performing schemata in the population can propagate at nearly the same rate as a GA without immigration. In contrast, if the population size is increased to enhance exploration, the schemata propagate more slowly, due to decreased selection pressure on good schemata. Therefore, immigration increases exploration while maintaining selection pressure (exploitation).

3. When a GA operates on a deceptive function [71] low-order schemata that are present in the optimal string have poor average fitness values. Individuals containing these schemata perform poorly, and are replaced in the population during the selection and recombination process. Immigration provides the GA with repeated opportunities to acquire optimal building blocks, even after they have been discarded.

4. The inclusion of an immigration operator does force a tradeoff in the GA. Immigration exchanges poor performers with random members. Each random member must be evaluated, which may increase the number of evaluations required to find the function optimum; however, immigrants can also bring missing structure to the population which should reduce the number of evaluations required to find the optimum. Therefore, the tradeoff of increased evaluations versus increased structure must be examined experimentally to determine the applicability of immigration in a GA.

One way to look at a GA modified with the immigration operator is as a GA with a large "virtual" population that maintains much of the selection pressure of a smaller population.

4.3.4 The Implementation of Two Genetic Algorithms

In this study, two different GA's were implemented, each with and without immigration. The first algorithm is a steady state GA. Each iteration of the algorithm we used is described as follows.

Steady State GA

1. Evaluate each new member of the population and assign a fitness value.
2. Probabilistically select two members from the population based on fitness. These members are parents.
3. Perform one-point crossover on the parents at a random string position to form two children.
4. Probabilistically perform mutation on the children.
5. Replace the two worst members of the population with the children.
6. Probabilistically perform mutation on the rest of the population (optional).

The algorithm is modified to include immigration by the addition of the following step.

1.5. Generate and evaluate m random members and replace the m worst members of the population with the m random members.

The second algorithm is based on Goldberg's restarted GA. Given a population of size n, each iteration of the algorithm proceeded as follows.

Restarted GA

1. Evaluate each member of the population and assign a fitness value.
2. Compute the bitwise convergence of the population.
3. If the convergence is greater than a given threshold, replace all but the best two members of the population with randomly generated and evaluated members.
4. Probabilistically select n - 2 members from the population based on fitness.
5. Randomly order the n - 2 selected individuals and form pairs. These pairs are parents.
6. Perform one-point crossover on each set of parents at a random string position to form children.
7. Replace the n - 2 worst members of the population with the children.
8. Probabilistically perform mutation on the children.

For selection, the restarted GA used Stochastic Universal Sampling as described by Baker [76].

The algorithm is modified to include immigration by the addition of the following step:

1.5. Generate and evaluate m random members and replace the m worst members of the population with the m random members.

Only n -2 members are selected each generation to insure survival of the best two performing members.

4.3.5 Test Suite of Functions

As stated earlier, immigration imposes a tradeoff in a GA. The added structure introduced by the random members occurs at the cost of evaluating each random member immigrated. Given this tradeoff, the type of functions where immigration should achieve a favorable balance between these factors and increase performance of the steady state GA are those functions with local optima that are difficult to avoid or escape. Functions of this nature require the steady state GA to be more circumspect while converging, and therefore may require sampling more structure. Immigration introduces the needed structure that may have been discarded from the population. By increasing the structure available to the GA, immigration should reduce the chance of converging at a local optimum, and thereby reduce the overall number of function evaluations by the steady state GA.

On the other hand, on functions that are unimodal, a steady state GA with immigration should perform poorly. A steady state GA operating on a unimodal function has a reduced chance of losing the structure necessary to find the global optimum. Adding the immigration operator introduces redundant structure at the cost of function evaluations. In this case, the tradeoff between added structure vs. added evaluations does not achieve a favorable balance, since the added structure would already be in the population. Therefore, a steady state GA with immigration should increase the number of function evaluations required to find the global optimum of a unimodal function.

It is difficult to predict the effect of immigration on a restarted GA. Like the steady state GA, immigration should allow a small population GA to sample more structure, and help prevent the GA from settling into a local optimum from which it

is difficult to escape. This should reduce the number of function evaluations required by the GA for multimodal functions.

For unimodal functions, it is reasonable to believe that the small population must converge a number of times before the global optimum of the function is found. Each convergence imports a host of new random members. It is difficult to predict whether importing random members after convergence is more efficient than immigrating random members during convergence, since both techniques maintain a high rate of schema processing.

As described in section 4.1, the shape of the ARM energy function is highly nonlinear, and possesses many local minima. Since it is difficult to describe the shape of the ARM function for a given asserted effect, a more understandable set of functions have been created to point out the types of energy surfaces that the GA might expect. The use of these functions also demonstrates that the immigration operator is applicable to functions other than the energy of the ARM model.

The experimental suite consists of a set of six functions that have different types of local and global optima. The suite was created to examine the ability of a GA to escape or avoid difficult local optima. This is reflected in the number of function evaluations required to find the global optimum of the function. By using a suite of this nature, one can determine the type of problems that prove difficult for a GA to solve, and show how the immigration operator affects performance.

Each of the functions is defined on a 20 bit binary string. The optimum cost value for each function is 0.0, which is the minimum value of the function. The functions are described as follows:

1. F1 (ODDEVEN): The purpose of this function is determine how well a GA can combine low-order, high-performing schemata into a structure where good local performance may lead to poor global fitness. In terms of the ARM, this is similar to subsets of agent symbols that work well together, but work poorly

when combined into larger sets due to higher order relationships.

A sliding window of length 4 is moved one bit at a time over a twenty bit member. The maximum cost is assigned 17.0. Each time the pattern 0101 or 1010 appeared in the window, 1.0 is subtracted from the cost. A pattern of alternating 1's and 0's, 01010101010101010101 or its complement produces the minimum cost of 0.0. A string of all 1's or all 0's has the maximum cost of 17.0.

For example, the string 01010111111111111111 has a functional value of 17.0 - 3.0 = 14.0 since there are three 4 bit strings of alternating patterns. The first one starts at position 0 and is 0101, the second begins at position 1 and is 1010 and the third begins at position 2 and is 0101.

This function contains local minima (optima) that may trap the GA. Consider the member 01010101011010101010. This member has a cost of 17.0 - 14.0 = 3.0 that would indicate that it is a near optimal member; however, the member must actually invert the values of 10 consecutive bits in order to achieve the the optimal configuration, which is a large Hamming distance. Such large Hamming distance disturbances are difficult to create through the mutation operator without destroying the good structure in the population. Therefore, if the population converged around this pattern or a similar one, the GA would be trapped in a local optimum. The example demonstrates that the combination of low-order, high-performing building blocks may produce a string that is far from the global optimum.

2. F2 (DECEPT1): The purpose of this function is to determine the performance of a GA on a difficult, two bit deceptive problem. For the ARM, this function determines if a GA can escape a robotic action that forms a good local minimum in the energy function, to find a very different robotic action that is the

optimal solution.

The 20 bit member represents four non-overlapping fields, each of length 5 bits. The maximum cost is assigned 28.0. For each five bit field, 1.2 is subtracted from the maximum cost for each bit in the field that is a 1. However, if all five bits in the field are 0, 7.0 is subtracted from the maximum cost. Therefore, a field of five 1 bits subtracts 6.0 from the maximum cost, so the member 11111111111111111111 has a cost of 4.0. The minimum cost member is all 0's, and has a cost of 0.0.

For example, the member 00000111111010100001 subtracts 7.0 for the maximum cost for bits 0-4, 6.0 for bits 5-9, 3.6 for bits 10-14 and 1.2 for bits 15-20, for a total cost of 28.0 - 17.8 = 10.2

This function falls within the class of GA-hard problems [71], since it contains low-order deceptive schemata. These indicate that the function minimum is a string of all 1 bits when it is really a string of all 0 bits. In terms of average member fitness, this function can be described as:

- f(*...*0*...*) < f(*...*1*...*)
- f(*...*00*...*) < f(*...*01*...*), f(*...*10*...*) < f(*...*11*...*)
- Also, f(*...*000*...*) < f(*...*111*...*)

3. F3 (DECEPT2): The purpose of this function is to determine the performance of a GA on a simpler, one bit deceptive problem. For the ARM, the same comparison is valid.

   A sliding window of length 4 is moved one bit at a time over the twenty bit member. The maximum cost is assigned 51.0. At a given window location, each 1 bit in the window subtracts 0.5 from the maximum value. If all bits in the window are 0, 3.0 is subtracted from the maximum value. The minimum

cost of the function is 0.0 and occurs when each bit in the individual is 0. When each bit in the individual is 1, the cost is 17.0. For example, the string 11110000111111111111 would have a cost of 22.0.

This function also contains low-order deceptive schemata, that indicate the function minimum is a string of all 1 bits, when it is really a string of all 0 bits. In terms of average member fitness, this function can be described as:

- f(*...*0*...*) < f(*...*1*...*)
- f(*...*00*...*) < f(*...*11*...*)

4. F4 (MIRROR): The purpose of this function is to examine the ability of a GA to process high-performing, high-order schemata while maintaining the consistency of low-order schemata. The cost function is designed to be much more sensitive to the high-order schemata than the low-order schemata. For the ARM, this function tests if the GA can throw away small subsets of symbols in a robotic action that perform well, to find ones that perform better.

   A maximum cost of 39.0 is assigned. Bit 19 of the member is made contiguous to bit 0 for wrap around. For each bit that is the same as its rightmost neighbor, 0.5 is subtracted from the cost. Also, if bit i ($0 \leq i \leq 9$) and bit $i + 10$ differ, 3.0 is subtracted from the cost (e.g., if bit 1 is different from bit 11, 3.0 is subtracted, if bit 2 is different from bit 12, 3.0 is subtracted, etc.). The minimum cost of 0.0 occurs when a string of ten 1's is followed by a string of ten 0's. Since wrap around is allowed, the pattern can begin anywhere in the member. A string of 1's or all 0's has a cost of 20.0

   For example, 00001111111111000000 has cost 0.0 since ten 1's are followed by ten 0's. The string 10101010101010101010 has a cost of 9.0.

   This function should prove difficult for the GA to solve. Consider the member 00011100001110001111. It has a cost of 2.0 yet it is Hamming distance 6 away

from the optimal solution. This forms a local optimum from which it is very difficult to escape.

For this function, the major reduction in cost occurs when high-order schemata are consistent, with a slight reduction when low-order schemata are consistent. As shown in the example, this can lead to members that have strong high-order consistency, but poor low-order consistency. This creates local minima that prove difficult for a GA to avoid or escape.

5. F5 (EIGHTAWAY): This function tests a GA's ability to assemble schemata that are of different order. The function is more sensitive to higher-order schemata than it is to low-order schemata. For the ARM, this is similar to the F4, but is more difficult.

   A maximum cost of 41.0 is assigned. For each bit i in the member different from bit $i + 1$, subtract 0.5 from the cost. Also, the cost is reduced as follows.

   (a) For $(0 \leq i \leq 4)$

      i. Let $m = i$, let $n = m + 8$

      ii. Repeat

         A. If bit m is different than bit n subtract 2.0 from the cost.

         B. Let $m = n$. Let $n = m + 8$

         C. If $n \geq 20$, $n = n - 20$

      iii. Until $n = i$

   The minimum of this function occurs at 01011010101001010101 or its complement. A member of all 0's or all 1's has the maximum value of 41.0

   This function forces schema consistency for defining lengths 4, 8, 12 and 16. The function is very sensitive to these high-order building blocks. Consistency should also be maintained for low-order schemata, but must be violated in

some instances in order to achieve the optimal string. The local optima this function possesses are similar in nature to those possessed by the function F4 (MIRROR).

6. F6 (ONEMAX): This is the same bit counting function described by Ackley [90] and is unimodal. For the ARM, this tests the ability of the GA to search a very simple, unimodal energy function.

   A maximum cost is assigned 20.0. Each 1 bit subtracts 1.0 from the cost. The minimum cost occurs when all 20 bits are 1 and has a cost of 0.0.

4.3.6 Description of Experiments

The focus of the experiments is to determine the effect of immigration in a Genetic Algorithm on the test suite of functions. As stated earlier, the immigration operator should increase the exploration of a GA without significantly decreasing the exploitation of the GA.

For the steady state GA, the increased exploration should prevent the GA from prematurely converging and becoming trapped in a local minima. This fact should be reflected in the number of GA trials that require an excessively long time to find the function optimum. Also, since the population size is maintained, the GA should have the exploration power of a larger population with the exploitation of a smaller one. This would be reflected in the average number of function evaluations required to find the global optimum. Therefore, for the steady state GA, two performance criteria are examined.

1. The average number of evaluations required to find the function optimum is the first criteria.

2. The number of trials that required an excessive number of evaluations to find the function optimum is the second criteria. These trials are called "outliers."

For the restarted GA, the inherent reinitialization process should prevent the population from becoming trapped in a local optimum for many generations. Becoming trapped in a local optima leads to an excessive number of evaluations, or outliers. Since the restarted GA should prevent this, immigration should not significantly reduce the number of outliers for an experiment. However, immigration does increase the amount of exploration performed by the GA while the population is converging. The increased exploration may allow the GA to be more circumspect and avoid local minima. Avoiding local minima should decrease function evaluations. Therefore, if immigration aids this algorithm, it would be reflected in the average number of function evaluations required to find the optimum.

4.3.6.1 Design of the steady state GA experiment

Since exploration vs. exploitation is the focus of this work, each function is evaluated over a range of population sizes. The smallest population size is 30. For each function, the population sizes are repeatedly incremented by 10 members until the GA performs worse than with the previous population size.

The fitness function first ranks each member of the size n population. Then, a fitness value is assigned to each member s using the equation

$$Fitness(s) = exp(3\frac{n - rank(s)}{n}) \tag{4.6}$$

The above fitness assignment curve maps the best member to fitness value 20.0 and the worst member to fitness value 1.0. Using these exponential constants, the best 50 percent of the population has a ratio of 4.5:1 in fitness values. An exponential curve is used to accentuate better performing members while assigning similar fitnesses to poor performers. The curve also prevents high-performing individuals from taking over the population entirely, so the fitness function is not unduly sensitive to cost values. Davis [91] has also used ranked exponential fitness assignment.

The immigration rate m (number of random individuals immigrated each generation) in the experiments ranges from 0 to 4 individuals per generation. Also, a one-point crossover scheme is used.

The probability of child mutation is fixed at 0.005 mutations/bit. Early experimentation determined that the performance of the steady state GA improved when members of the population other than the current children were allowed to mutate. This seemed most important as the population began converging, so a dynamic mutation rate as a function of convergence is used. The dynamic population mutation rate is given by the equation

$$Mutations/Bit = 0.015(C - 0.5) \tag{4.7}$$

(where C is the bitwise convergence percentage of the population and ranged from 0.5 to 1.0).

Again, each population member is 20 bits long. Each experiment is assigned a population size and immigration rate, and is tested with 500 separate GA trials, each with a unique random population. Each GA trial counts the number of function evaluations until the global optimum is found. The maximum number of evaluations allowed per trial is 50000. This is performed on all functions in the test suite.

4.3.6.2 Design of the restarted GA experiment.

For this experiment, the population sizes begin at $n = 14$. For each function, the population size is increased by 2 members until the GA performs worse than with the previous population size.

The fitness function first ranks each member of the size n population. Then, a fitness value is assigned to each member s using the equation

$$Fitness(s) = exp(1.5\frac{n - rank(s)}{n}) \tag{4.8}$$

which assigns fitness values between 1.0 and 4.5. A smaller exponential constant (1.5 instead of 3.0) is chosen for this small population GA. This provides most of the population members with some chance to compete. However, this fitness scheme does enforce a 4.5:1 fitness ratio between the best and worst members of the population.

The immigration rate m in the experiments ranges from 0 to 4 individuals per generation. Also, a one-point crossover scheme is used. The probability of mutation is set at 0.005 mutations/bit.

For functions F1 - F5, the threshold convergence ratio is set 0.85. In other words, when the population is 85 percent bitwise converged, the two best members are kept and random members fill the remainder of the population. This convergence ratio was selected after some experimentation with values of 0.75 and 0.95. In general, for functions F1 - F5, a convergence value of 0.75 forced the GA to import random members before good structure had been developed, and led to an increased number of function evaluations. A convergence value of 0.95 often required convergence of members to a local optimum which was already present in the population. This also led to an excessive number of function evaluations.

For F6, a unimodal function, a convergence ratio of 0.95 provides the best performance. For this function, the population could not settle into a local optimum and could continue useful schema processing to a higher degree of convergence.

As with the steady state GA, each population member is 20 bits long. The same experimental constraints are also present.

4.3.7 Experimental Results

The experiments provided a measure of difficulty for each of the six functions in the suite. Figures 4.6 - 4.11 present a comparison of the performance of the steady state GA with immigration (dashed bars) and without immigration (solid bars) on

functions F1 - F6 of the test suite. Figures 4.12 - 4.17 present a comparison of the performance of the restarted GA. These plots reflect the average number of evaluations required by the GA to find the optimum value of each function. The number of immigrations per generation that produced these results is labeled in each figure.

Based on the experimental results, both GA's easily solved function F6 (ONE-MAX) which is a unimodal function. This was to be expected. Functions F1 (ODD-EVEN) and F3 (DECEPT2) proved only a little more difficult to the GA's. This indicates that the GA does a good job assembling low-order optimal schemata into an optimal string. It also indicates that the GA can overcome some deception in its search.

Function F4 (MIRROR) was next in level of difficulty, followed at a distance by F5 (EIGHTAWAY). Both of these functions required the development of high-order schemata and the consistency of low-order building blocks. The difficulty of the GA in achieving the global optimum of each function may be due to the crossover operation used. The strength of one-point crossover is its ability to assemble low-order building blocks into optimal strings. High-order schemata have a greater chance of being destroyed, as was demonstrated by these experiments. Perhaps the performance of the GA's would improve using a crossover mechanism that is less positionally biased.

Function F2 (DECEPT1), a two-bit deceptive function, proved extremely difficult to both GA's. When compared to function F3, a one-bit deceptive function, one can see that increased deception has a profound effect on the optimization capabilities of a GA.

The next sections describe in detail the experimental results, and examine effects of immigration on a Genetic Algorithm.

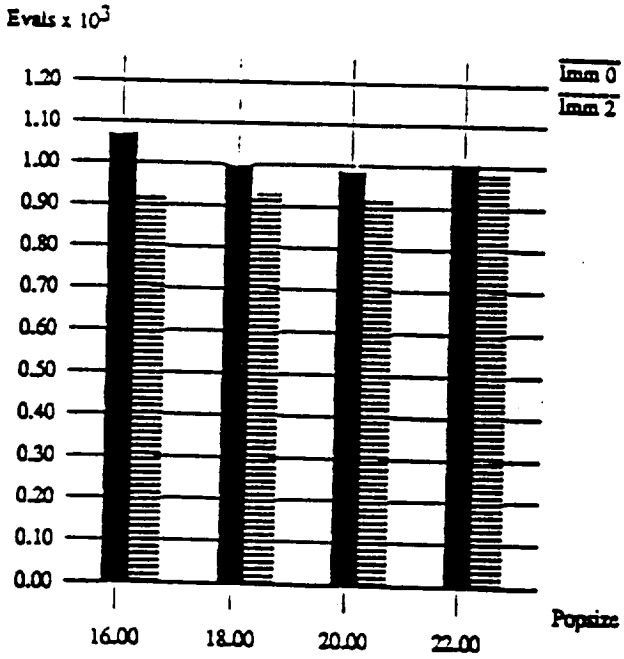


Figure 4.6: F1 - Average Number of Evaluations using Steady State GA

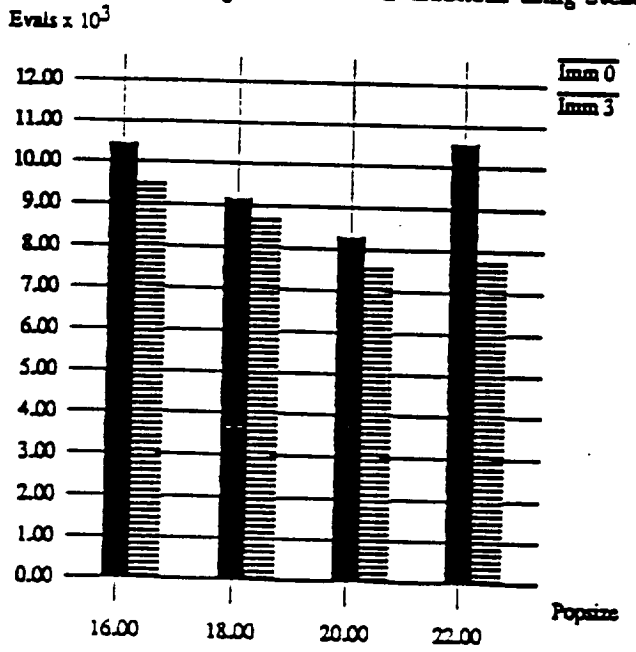


Figure 4.7: F2 - Average Number of Evaluations using Steady State GA

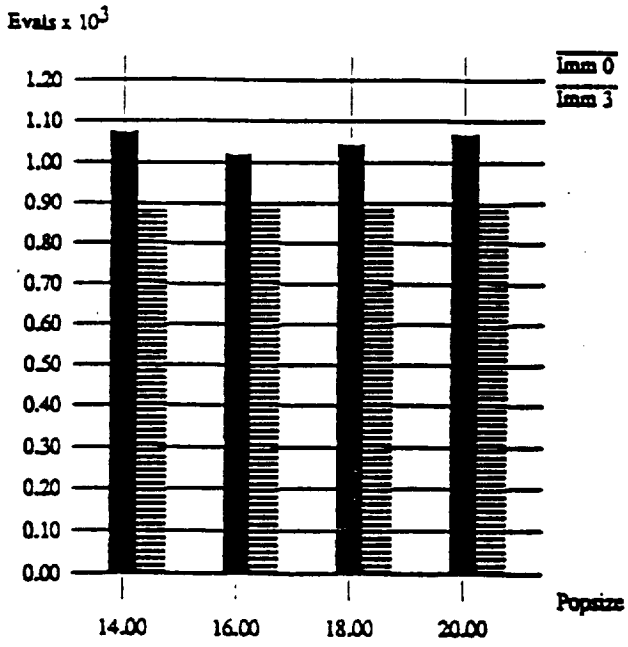


Figure 4.8: F3 - Average Number of Evaluations using Steady State GA

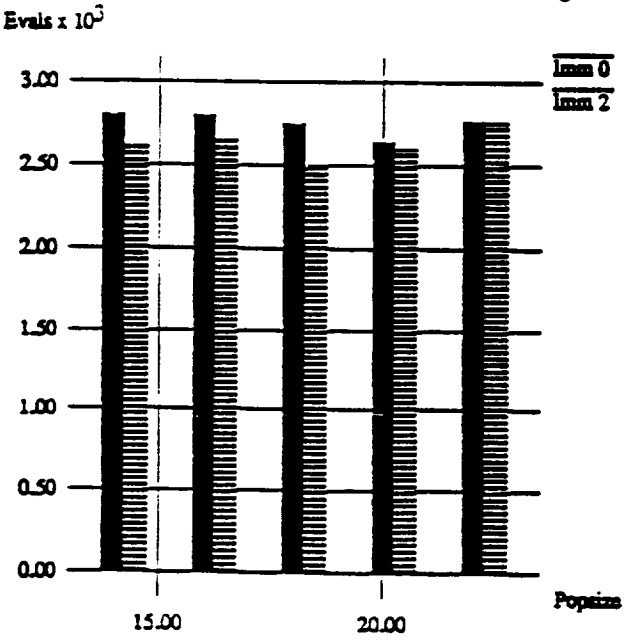


Figure 4.9: F4 - Average Number of Evaluations using Steady State GA

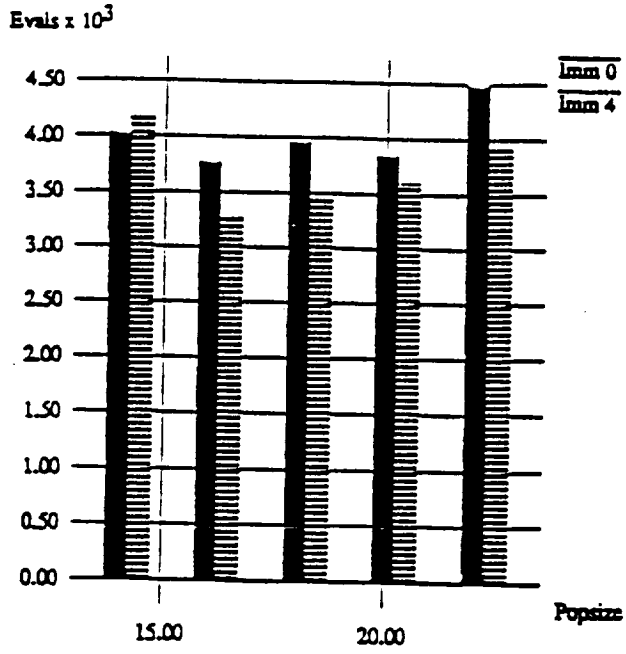


Figure 4.10: F5 - Average Number of Evaluations using Steady State GA

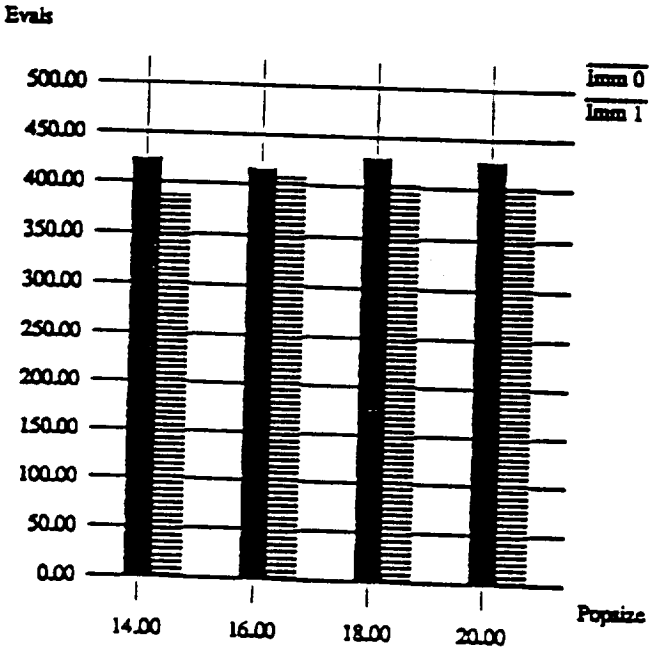


Figure 4.11: F6 - Average Number of Evaluations using Steady State GA

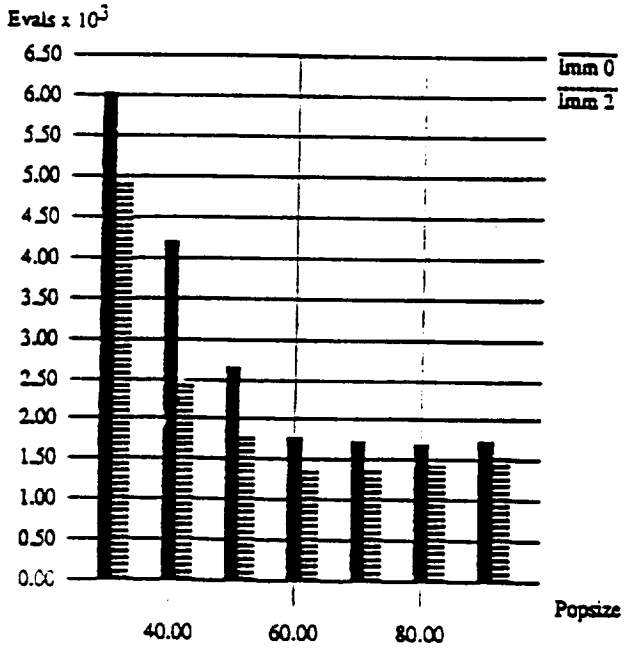


Figure 4.12: F1 - Average Number of Evaluations using Restarted GA

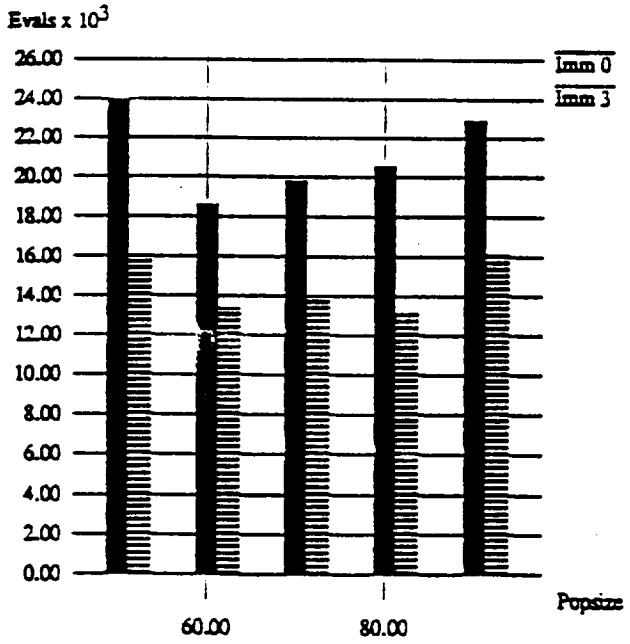


Figure 4.13: F2 - Average Number of Evaluations using Restarted GA

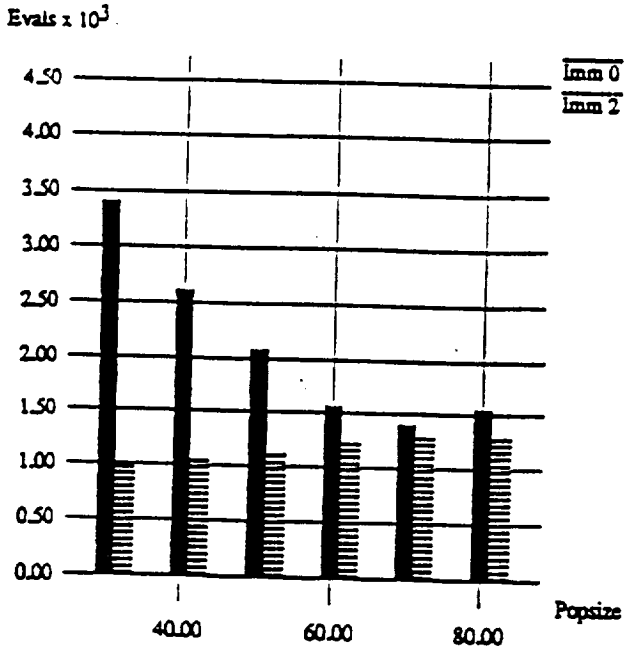


Figure 4.14: F3 - Average Number of Evaluations using Restarted GA

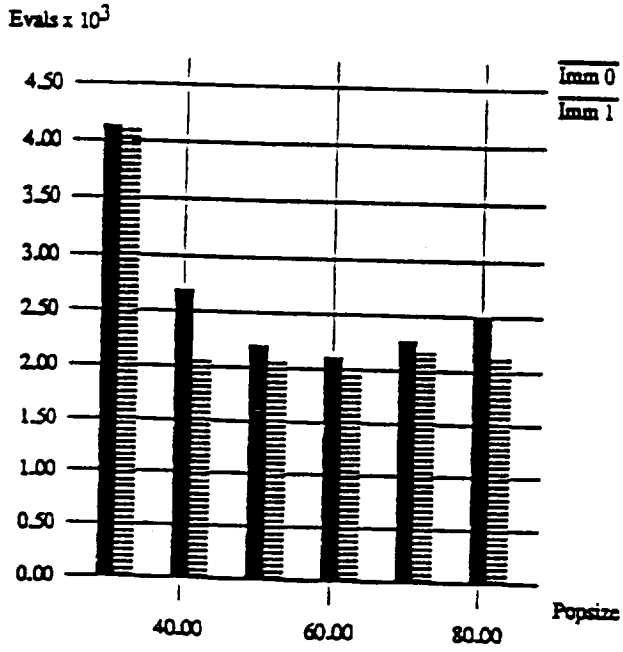


Figure 4.15: F4 - Average Number of Evaluations using Restarted GA

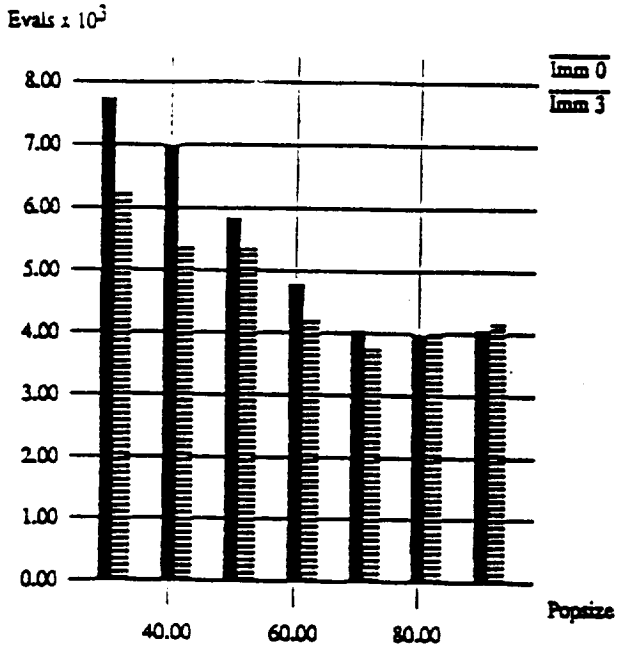


Figure 4.16: F5 - Average Number of Evaluations using Restarted GA

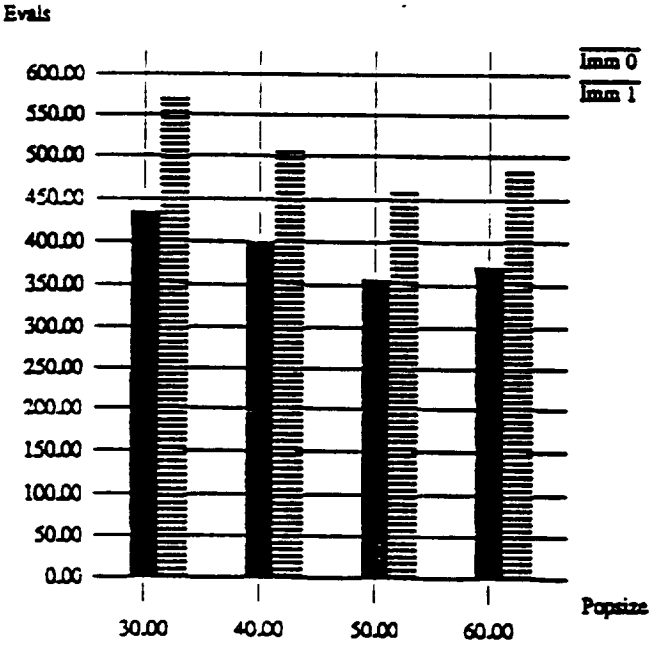


Figure 4.17: F6 - Average Number of Evaluations using Restarted GA

4.3.7.1 Steady state GA

The smallest average number of evaluations for functions F1 - F5 occurred when immigration was present. This was expected, since these functions contain many local optima. For function F6, the steady state GA without immigration outperformed the modified GA. Again, this was predicted, since F6 is a unimodal function.

It is important to note that these figures provide the best results of the steady state GA with immigration. The immigration rate that performed best for a function is called the "optimal" immigration rate for that function. In general, all immigration rates (greater than 0) up to and including the optimal rate resulted in an improvement in performance over the non-modified GA.

For function F1, (Figure 4.6) the GA without immigration produced the best results at a population size of 80 and required an average of 1688 function evaluations to find the optimum solution. With 2 immigrations per generation at a population size of 60, the GA required only 1352 function evaluations. Therefore, the GA without immigration resulted in a 24.8 percent increase in search time over the GA with immigration. In fact for each function F1 - F5, *immigration resulted in a reduction in the number of function evaluations.*

Examining Figures 4.6 - 4.11, it is interesting to note that for most of these functions, the reduction in function evaluations using immigration is largest for small populations, and decreases as the population size increases. This indicates that the smaller populations require the added exploration that immigration provides, while larger populations possess sufficient exploration power.

Also, in most trials shown in these figures, the GA with immigration performed better than a GA with 10 more population members without immigration. This provides more evidence that immigration allows the GA to search the space of a larger population.

Further, in F1, F3 and F5 the optimum with immigration occurs in smaller populations than the optimum without immigration. This lends credence to the theory that immigration allows small populations to retain their selection pressure. This is demonstrated further in Figures 4.18 - 4.29.

The histograms in Figures 4.18 - 4.29 present the results of 500 GA trials on each function with and without immigration. The figures present the number of trials (Y axis) that require a given number of function evaluations (X axis) to find the global optimum for a range of population sizes. The different population sizes are represented by solid, dashed and dotted lines. Figures 4.18, 4.20, 4.22, 4.24, 4.26, and 4.28 show results of the GA without immigration. Figures 4.19, 4.21, 4.23, 4.25, 4.27, and 4.29 show the results with optimal immigration. For example, in Figure 4.18, the GA without immigration and a population size of 40 (solid line) found the optimal solution in 200 function evaluations (X axis) in 129 out of its 500 separate trials for function F1.

To reduce the length of the X axis, all trials that require an excessive number of function evaluations are grouped together at the last point on the X axis. For example, in Figure 4.18, the GA with 0 immigrations and a population size of 40 had 78 points that required more than 5000 evaluations to find the optimal solution. These points are referred to as "outliers."

Examining the results of the GA without immigration, one can see that the peak generally shifts to the right with increasing population size. This demonstrates the decrease in selection pressure, which inhibits the GA from finding easy solutions quickly. From these figures, one also notes that increasing population size reduces the number of outliers. This indicates that an increase in population size increases the exploration power of the GA.

Let us now compare the GA without immigration to the GA with immigration. As shown in Figures 4.18 and 4.19, the GA with immigration has significantly fewer

outliers than the GA without immigration This is an example of how immigration can increase exploration. Further, the peaks of Figure 4.19 occur at about the same number of function evaluations (X axis) as the peaks in Figure 4.18. As discussed above, if selection pressure was decreased by immigration, we could expect the peaks in Figure 4.19 to be shifted right of the peaks in Figure 4.18. This is not the case, so *the GA with immigration maintains selection pressure and exploitation power.*

Figures 4.20 - 4.23 present similar results. Further, for these functions the magnitude of the peaks actually increased with immigration. This is due to the reduction in the number of function evaluations for trials to the right of the peaks, another indication of increased exploration.

Figures 4.24 - 4.27 the peaks occurring near the same X axis location, but again show that immigration does not always eradicate all the outliers for various population sizes. It does show, however, that immigration still reduces the number of outlying trials. Since these outliers contribute heavily to the average number of function evaluations required to find the optimum, it is clear that eliminating outliers reduces this value.

Figures 4.28 and 4.29 show the result of function F6 with no immigrations and with 1 immigration per generation. These experiments verified our prediction that a steady state GA with immigration would perform poorly on a unimodal function. As shown by the plot, immigration shifted the peak to the right and reduced it. The average number of evaluations rose from 358 (without immigration) to 461 (with immigration). In this case, immigration did not achieve the balance between added evaluations and missing population structure. This indicates that immigration is not necessary for functions in which the structure can be selected reliably and propagated easily through the population of a steady state GA.

Overall, these experiments show that a population size between 60 and 70

members performs best for the steady state GA described above. To improve performance, an immigration rate of 2 or 3 members per generation should be used on functions that contain difficult local optima.

4.3.7.2 Restarted GA

As with the steady state GA, the smallest average number of evaluations for functions F1 - F5 occurred when immigration was present in the restarted GA. This was predicted, since these functions contain many local optima.

However, for function F6, the unimodal function, the restarted GA with immigration outperformed the GA without immigration. This indicates that adding random members during convergence improves the efficiency of a restarted GA over both unimodal and mutimodal functions. This phenomena is quite interesting, and should be studied in further detail.

Again, these figures provide the best results of the restarted GA with immigration. In general, all immigration rates (greater than 0) up to and including the optimal rate resulted in an improvement in performance over the non-modified GA.

These experiments show that a population size between 16 and 20 members performs best for the restarted GA described above. To improve performance, an immigration rate of 2 or 3 members per generation should be used on functions that contain difficult local optima. Searching unimodal functions is more efficient at smaller populations, and can also benefit from the effects of immigration.

4.3.7.3 Immigration: Conclusions and recommendations

This study has examined the tradeoff between exploration and exploitation in Genetic Algorithms. It conjectured that a GA can increase exploration power while maintaining selection pressure by replacing poor performing individuals in a population with random members.

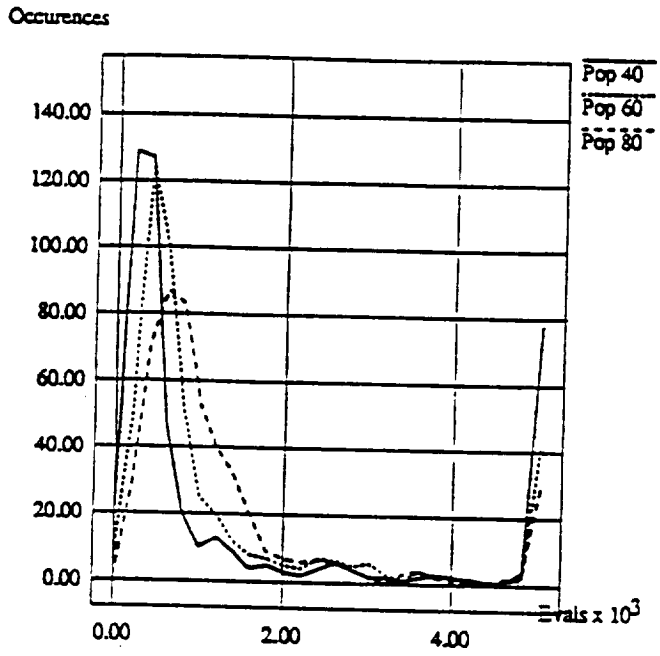


Figure 4.18: F1 - 0 Immigrations Per Generation

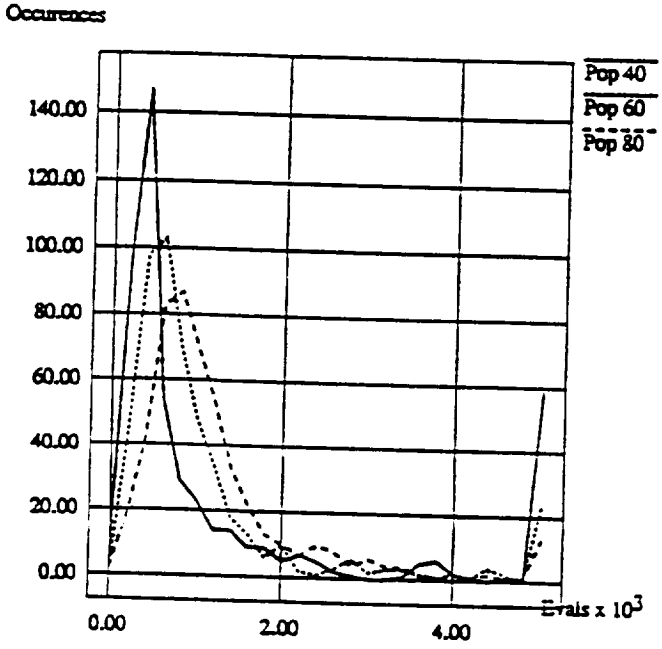


Figure 4.19: F1 - 2 Immigrations Per Generation

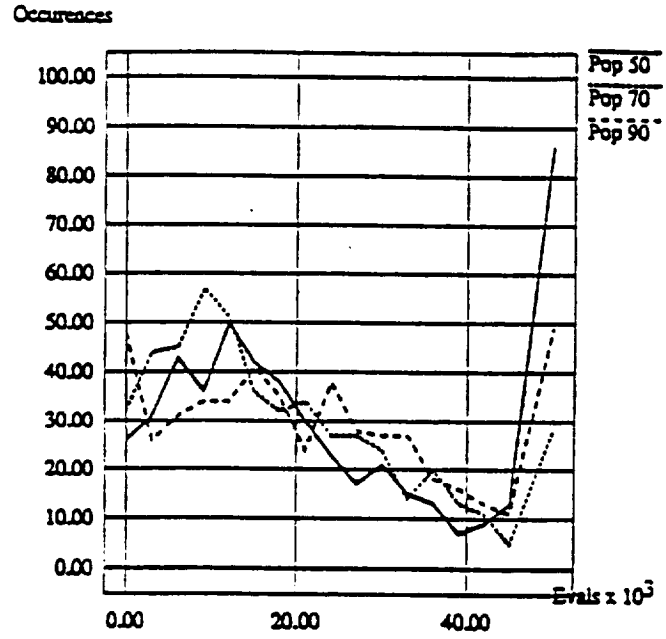


Figure 4.20: F2 - 0 Immigrations Per Generation

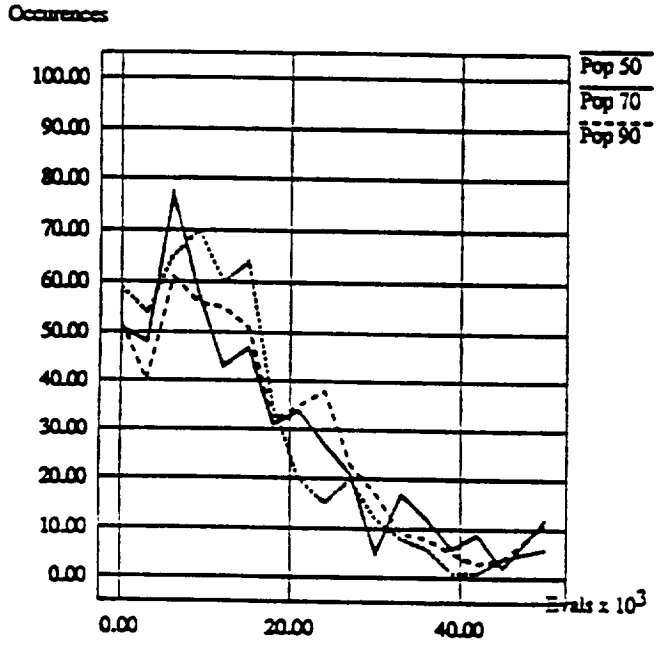


Figure 4.21: F2 - 3 Immigrations Per Generation

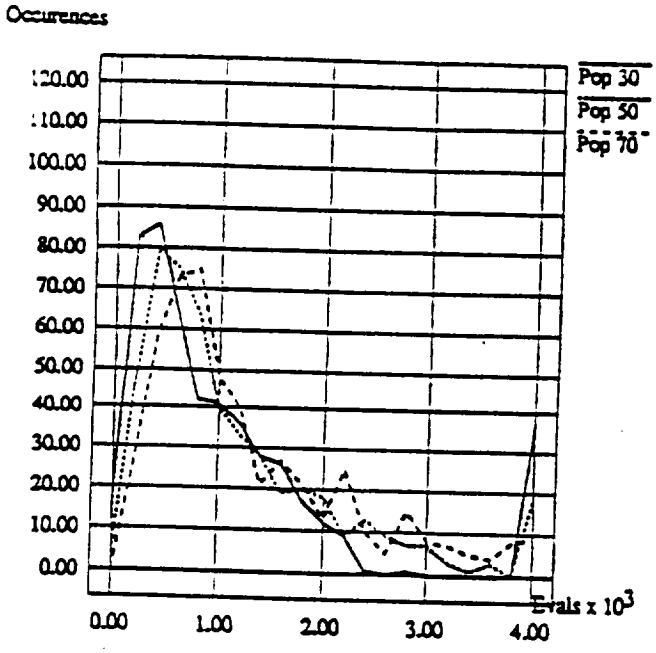


Figure 4.22: F3 - 0 Immigrations Per Generation

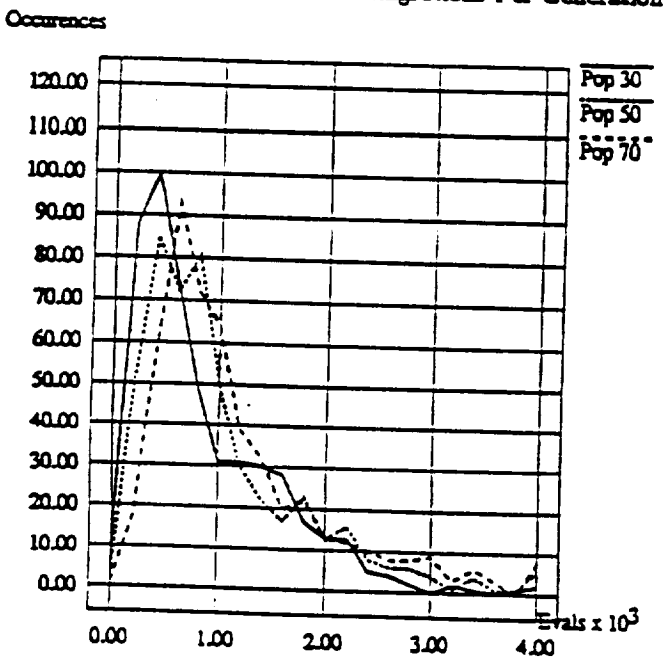


Figure 4.23: F3 - 2 Immigrations Per Generation

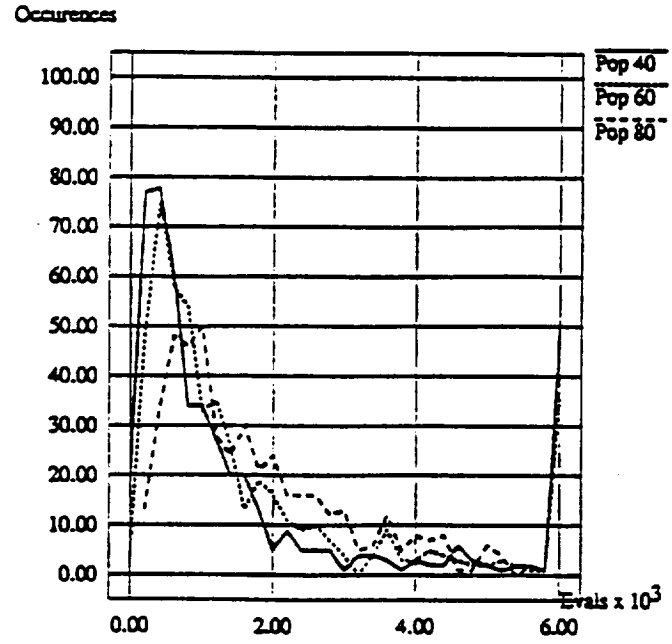


Figure 4.24: F4 - 0 Immigrations Per Generation

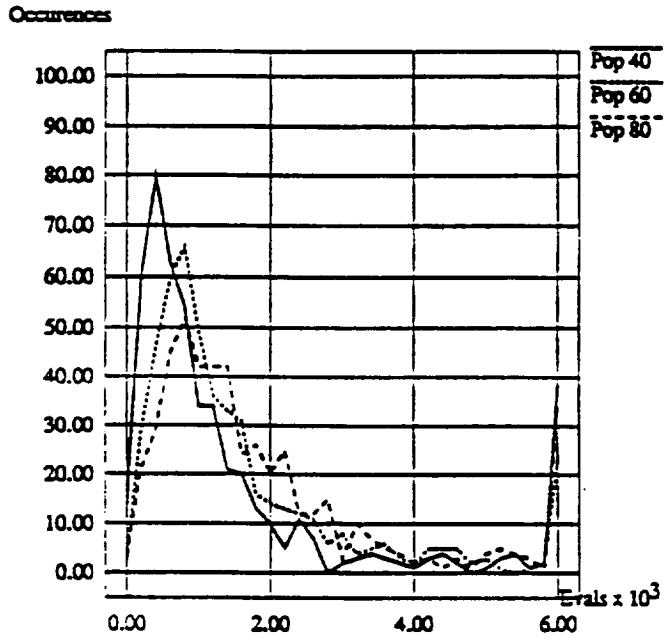


Figure 4.25: F4 - 1 Immigrations Per Generation

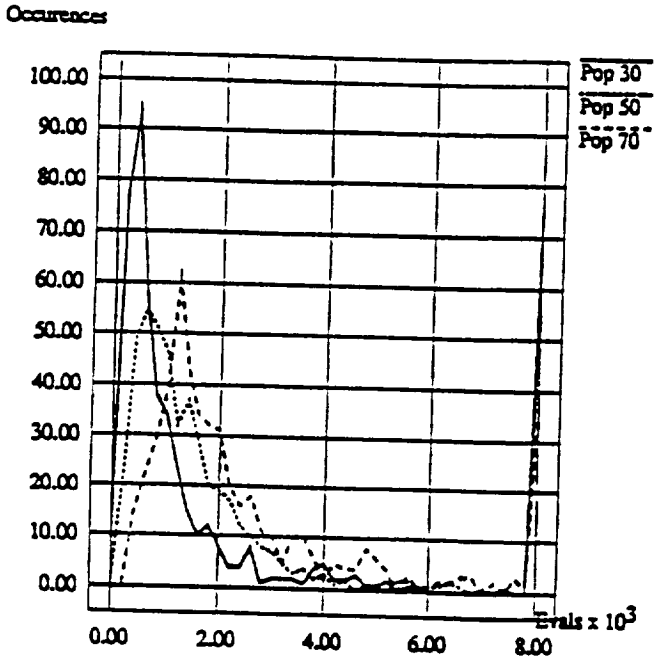


Figure 4.26: F5 - 0 Immigrations Per Generation

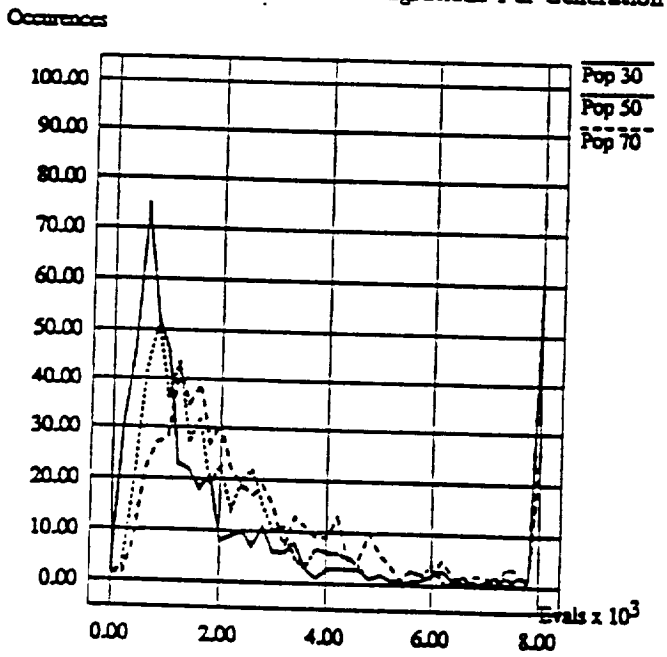


Figure 4.27: F5 - 3 Immigrations Per Generation

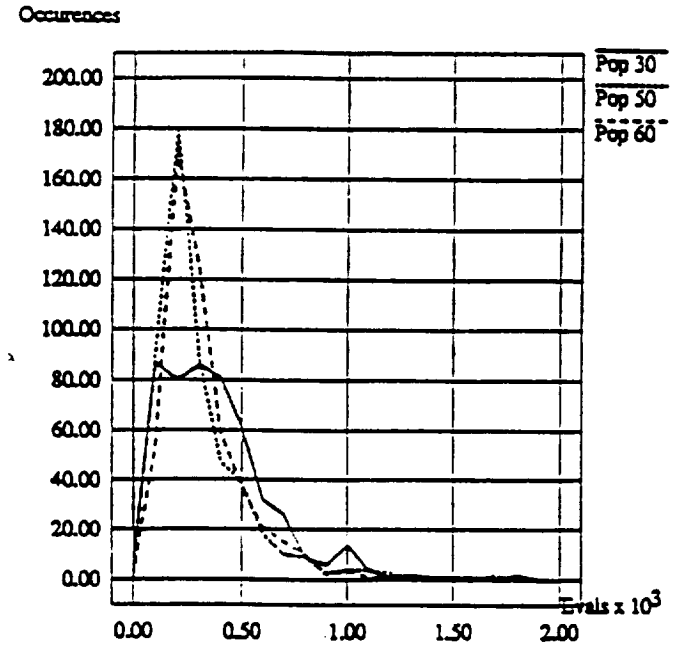


Figure 4.28: F6 - 0 Immigrations Per Generation

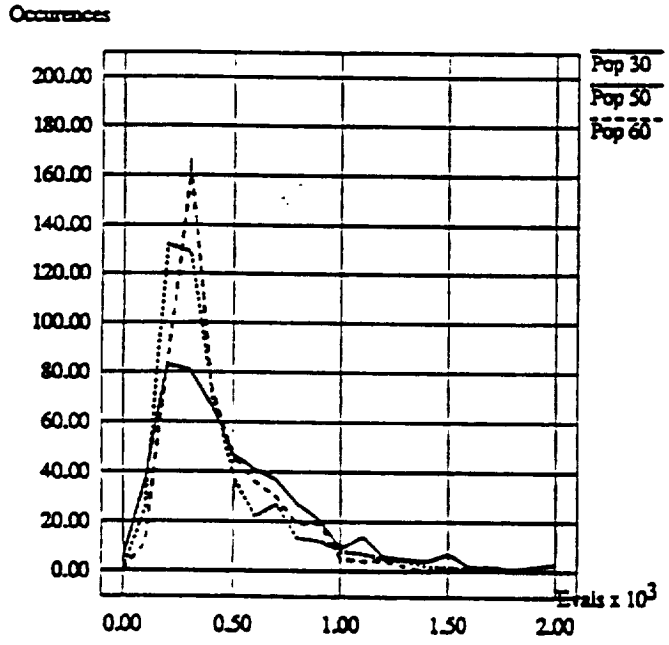


Figure 4.29: F6 - 1 Immigrations Per Generation

The results of the experimentation show that immigration improves the performance of steady state GA's when optimizing functions that contain local optima that are difficult to avoid or escape. The experiments also show that immigration improves performance for a spectrum of functions using a restarted GA.

Although these experiments are completed, there are several recommendations for future work. Testing immigration on a generational GA would be the next logical step of this research. Also, a further examination of immigration versus restart would be in order. Finally, testing functions F4 (MIRROR) and F5 (EIGHTAWAY) using a less positionally biased crossover operator, such as uniform crossover, would prove interesting.

In conclusion, this study demonstrates that immigration is a viable operator for improving the efficiency of GA's on difficult optimization problems.

4.4 Convergence of a GA using Immigration

In previous sections, the addition of the immigration operator to a Genetic Algorithm was shown to reduce the search time required to find the global optimum for a given class of functions. This section develops some theoretical underpinnings for the immigration operator, and proves that a GA enhanced with immigration will converge in probability to the global optimum of a given function.

One of the main problems of the Genetic Algorithm is premature convergence. If the entire GA population converges to the value of a single member before the global optimum is found, the selection and crossover operations will never be able to create the optimal member. In this case, only the mutation operator can bring new structure to the population; however, since mutation changes just a few individual bits, it only explores regions local to the converged population. If the global optimum is a large Hamming distance from where the population has converged, there is little chance that mutation will aid in discovering it. In fact, there is no proof of

convergence of a GA to the global optimum of a given function.

Convergence, in the GA sense, denotes the degree of uniformity of the members of the population. Convergence to a global optimum, in an optimization sense, means that the sequence of "best" points generated by the search algorithm tends to the optimal solution. For the rest of this section, "convergence" will take on the latter of these definitions.

By embedding the immigration operator into the GA, we can develop a proof that shows that the modified GA will converge in probability to the optimal function value. The initial idea for the proof is based on the concept of Spacer Steps, as presented by Luenberger [65]. The Spacer Step technique states that an algorithm that is known to converge to the global optimum of a cost function can be combined with another algorithm, which may not converge to the optimum, and the resulting algorithm will converge to the global optimum.

To develop the proof, we will first present an algorithm that converges in probability to the global optimum of a function. Then, we will illustrate the methodology of combining this algorithm with the GA using Spacer Steps, and relate this to immigration. The use of the Spacer Step technique serves to introduce the actual proof of convergence, which is based on a probabilistic argument.

First, let us consider a search algorithm called Random Search. Given a cost function Q, a state vector $\mathbf{X}_i = (x_1, x_2, \cdots, x_n)$ at iteration i, and a prespecified value ϵ, perform the following.

Random Search

1. Generate a random vector $\mathbf{X}^{(\mathbf{rand})}$
2. If $Q(\mathbf{X}^{(\mathbf{rand})}) < Q(\mathbf{X}_i) - \epsilon$ then set $\mathbf{X}_{i+1} = \mathbf{X}^{(\mathbf{rand})}$.
3. Else, set $\mathbf{X}_{i+1} = \mathbf{X}_i$.

This Random Search algorithm has been shown to converge in probability to the global minimum (optimum) of a cost function, Q [66, 67].

The technique of Spacer Steps is described by Luenberger as follows [65].

> Suppose B is an algorithm which together with the descent function Q and a solution set Γ, converges globally to the optimum of Q. Let us define another algorithm C by $C(\mathbf{X}) = \{\mathbf{Y} : Q(\mathbf{Y}) \leq Q(\mathbf{X})\}$. In other words, C applied to $\mathbf{X}$ can give any point so long as it does not increase the value of Q. Then, B represents the spacer step in the algorithm CB and the overall algorithm CB converges globally to the optimum of Q.

In the above description, B is the Random Search algorithm, which is known to converge in probability to the optimum of a function, and C is the Genetic Algorithm. To combine Random Search and the Genetic Algorithm, an operator must be developed that adds random members to the GA search. This operator has been developed and is called immigration. The only other constraint is that the GA cannot remove the current best performing member from the population. Hence, the GA modified with immigration as presented in earlier sections forms the CB algorithm.

The Spacer Step technique serves to illustrate the effect of adding Random Search to the GA in the form of an immigration operator. However, the Spacer Step technique has not been explicitly proven for probabilistic algorithms. Since Random Search falls into the class of probabilistic techniques, to be mathematically precise, we must construct another proof. This proof is similar to the one described by Matyas [66].

First, let us present the Random Search algorithm, combined with another search algorithm, A. The combined algorithm searches over a discrete domain of binary vectors.

Combined algorithm A and Random Search

Given a cost function Q, a state vector $\mathbf{X_i} = (x_1, x_2, \cdots, x_n)$ at iteration i, where $x_k \epsilon \{0, 1\}$, repeat the following.

1. Generate a random vector $\mathbf{X^{(rand)}}$
2. If $Q(\mathbf{X^{(rand)}}) < Q(\mathbf{X_i})$ then set $\mathbf{X_{i+1}} = \mathbf{X^{(rand)}}$.
3. Else, set $\mathbf{X_{i+1}} = \mathbf{X_i}$.
4. Generate a set of search points, $\{\mathbf{X^{(A)}}\}$ from algorithm A and select a member, $\mathbf{X^{(A)'}}$, of the set such that $\forall \mathbf{X} \epsilon \{\mathbf{X^{(A)}}\}$, $Q(\mathbf{X^{(A)'}}) \leq Q(\mathbf{X})$
5. If $Q(\mathbf{X^{(A)'}}) < Q(\mathbf{X_{i+1}})$ then set $\mathbf{X_{i+2}} = \mathbf{X^{(A)'}}$.
6. Else, set $\mathbf{X_{i+2}} = \mathbf{X_{i+1}}$.
7. If $Q(\mathbf{X_{i+2}})$ is not optimal, set $i = i + 2$ and goto 1.

This algorithm will converge in probability to the global minimum (optimum) of the cost function Q., i.e the sequence $\{\mathbf{X_0}, \mathbf{X_1}, \cdots, \mathbf{X_k}\}$ as $k \rightarrow \infty$ will tend to the optimal value of $\mathbf{X}$ over $Q(\cdot)$, which will be referred to as $\mathbf{X^{opt}}$

Theorem 4.1. The sequence $\{\mathbf{X_0}, \mathbf{X_1}, \cdots, \mathbf{X_k}\}$ provided by the combined algorithm A and Random Search will tend to $\mathbf{X^{opt}}$ as $k \rightarrow \infty$.

Proof:

Let us first define the set $G[k]$ as:

$$G[k] = \{\mathbf{X} : Q(\mathbf{X}) < k\} \quad . \tag{4.9}$$

This is the set of all points that have a cost less than k. The set $G[k]$ depends on the form of the function Q. We assume that Q is regular, in the sense that $G[k]$ has the following property:

Property A. $(\forall k)(k > Q(\mathbf{X^{opt}}), G[k]$ is a non-empty set.

We must show that $P(\mathbf{X_k} \neq \mathbf{X^{opt}}) \rightarrow 0$ as $k \rightarrow \infty$.

1. Let $\mathbf{X} \epsilon \{0,1\}^n$ This is the domain of the the combined search algorithm.

2. Let $f^{(rand)}(\mathbf{X})$ be the probability density function of the discrete random variable $\mathbf{X}$ generating $\mathbf{X^{(rand)}}$. We know that $\sum_{\mathbf{X}} f^{(rand)}(\mathbf{X}) = 1$. We guarantee that $(\forall \mathbf{X}) f^{(rand)}(\mathbf{X}) \geq 0$.

3. Let $f_{i+1}^{(A)}(\mathbf{X})$ be the probability density function of the discrete random variable $\mathbf{X}$ generating $\mathbf{X^{(A)}}$ at iteration $i+1$. We know that $\sum_{\mathbf{X}} f_{i+1}^{(A)}(\mathbf{X}) = 1$.

4. Let us define a *succesful step* from state $\mathbf{X}$ at iteration i as a step that generates $\mathbf{X_i}$ such that $Q(\mathbf{X_i}) < Q(\mathbf{X})$. In other words, it is a step that reduces the cost function, Q.

5. The probability $P_{Q_{i+1}}(\mathbf{X})$ of a successful step from state $\mathbf{X}$ at iteration i or $i+1$ can be expressed as:

$$P_{Q_{i+1}}(\mathbf{X}) = \sum_{G[Q(\mathbf{X})]} (f^{(rand)}(\mathbf{X}) + f_{i+1}^{(A)}(\mathbf{X})) - \sum_{G[Q(\mathbf{X})]} \sum_{G[Q(\mathbf{X})]} (f^{(rand)}(\mathbf{X}) f_{i+1}^{(A)}(\mathbf{X})) \tag{4.10}$$

6. By Property A, we know that $G[Q(\mathbf{X})]$ is non-empty if $\mathbf{X} \neq \mathbf{X^{opt}}$ and since $(\forall \mathbf{X})(\forall i)\ f^{(rand)}(\mathbf{X}) > 0$, there must exist an $\alpha > 0$ such that $P_{Q_{i+1}}(\mathbf{X}) \geq \alpha$.

7. Let us set

$$m = 2^n. \tag{4.11}$$

In the combined search algorithm, if at least m steps are successful, it is guaranteed that we have found the point $\mathbf{X^{opt}}$. This is true if no step in the combined algorithm increases the cost of the current best member from one iteration to the next. Let us define s_k as the number of successful steps occuring up to and including step k.

8. Consequently the probability that $\mathbf{X_k} \neq \mathbf{X^{opt}}$ is less than the probability that the number of successful steps does not exceed m, i.e.,

$$P(\mathbf{X_k} \neq \mathbf{X^{opt}}) \leq P(s_k < m) \tag{4.12}$$

9. Since $P_{Q_{i+1}}(\mathbf{X}) \geq \alpha$, for $\mathbf{X} \neq \mathbf{X^{opt}}$, we can bound this by a binomial probability distribution

$$P(s_k < m) < \sum_{i=0}^{m} \binom{k}{i} \alpha^i (1-\alpha)^{k-i} \tag{4.13}$$

where k is the number of steps taken. Further, when $k > 2m$ and $\alpha < 0.5$,

$$\sum_{i=0}^{m} \binom{k}{i} \alpha^i (1-\alpha)^{k-i} < (m+1)\binom{k}{m}(1-\alpha)^k =$$

$$\frac{m+1}{m!}k(k-1)(k-2)\cdots(k-m-1)(1-\alpha)^k < \frac{m+1}{m!}k^m(1-\alpha)^k \tag{4.14}$$

10. Consequently,

$$P(\mathbf{X_k} \neq \mathbf{X^{opt}}) < \frac{m+1}{m!}k^m(1-\alpha)^k \tag{4.15}$$

11. For any $\alpha > 0$, it is clear that:

$$\lim_{k \to \infty} k^m(1-\alpha)^k = 0 \tag{4.16}$$

12. Therefore, $P(\mathbf{X_k} \neq \mathbf{X^{opt}}) \to 0$ as $k \to \infty$.

Q.E.D.

Of course, the algorithm A is the Genetic Algorithm, under the guarantee that is does not remove the best current member at any step. It is easy to see that this proof can be extended to situations in which the immigration step occurs infrequently.

4.5 Representation of Nodes for Genetic Optimization

The Genetic Algorithm has been shown to be adept at optimizing difficult cost functions, such as the energy function for the ARM. Each member of the GA is represented by a string of binary values. For the ARM, the binary member of the GA population must map to the output nodes of the network, because the purpose of associative recall is to find the correct set of asserted output nodes that minimizes the energy of the network for the given asserted input nodes. In other words, the population maintained by the GA represents different configurations of output nodes, and the fitness of each member reflects the energy for that configuration, when asserted on the network.

The issue of representation involves the mapping of a member of the GA population onto the output nodes of the ARM. By choosing the appropriate representation for the output nodes, it is possible to reduce the search time of the GA.

For example, Caruana and Schaffer [92] find that using a Gray code representation for integer or real population members outperforms straight binary coding in numerical optimization problems. The use of a Gray code allows nearby integer or real search points to have similar representations as members of the GA population. The similar representations are characterized by having small Hamming distances. Under binary coding, nearby search points may have vastly different codings in the GA population. This prevents the GA from developing a sense of "continuity" in the domain of the problem and hinders local search. Hollstein [93] also advocates the use of Gray encodings for such problems.

Goldberg [71] offers two basic principles for choosing a GA representation. These are the Principle of Meaningful Building Blocks and the Principle of Minimal Alphabets.

The Principle of Meaningful Building Blocks is stated as follows.

> The user should select a coding so that short, low-order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions.

This principle is based on the one-point crossover operation, which has a strong positional bias towards maintaining short, low-order building blocks.

The Principle of Minimal Alphabets is stated as follows.

> The user should select the smallest alphabet that permits a natural expression of the problem.

This principle, Goldberg explains, is based on the idea that smaller alphabets provide greater numbers of schemata per bit of information. The lower limit of alphabets is the binary representation used in most Genetic Algorithms.

With this information, we can develop a suitable GA representation for output nodes of the ARM.

Recall from Chapter 3 that the structure of the specific rules provided to the ARM is

$$ACTOR\ ACTION\ OBJ_1\ OBJ_2\ \cdots\ OBJ_m\ \rightarrow$$

$$OBJ_1\ STATE\ OBJ_2$$

The first part of the rule is called the robotic action. The second part of the rule is the effect. The ARM is designed such that it is provided with a desired effect as input and recalls a robotic action as output. Therefore, the GA must represent the nodes corresponding to

$$ACTOR\ ACTION\ OBJ_1\ OBJ_2\ \cdots\ OBJ_m$$

Based on the design of the ARM, only one *ACTOR* node, one *ACTION* node, and one of each OBJ_x node can be asserted together, to maintain the structure of the rule grammar. This is a relatively sparse representation, given the number of output

nodes in the network. If each node was directly mapped to a single bit in each GA member, and each output level was n nodes long, each member would have length equivalent to the number of output nodes in the ARM, which is $n(m+2)$, leading to a search space of $2^{n(m+2)}$ points. Further, since the representation allows only one symbol to be asserted on each level of the ARM, most of the GA search would be spent removing population members (created through crossover and mutation) that violate this constraint.

A better representation is to binary encode the symbols. Each output level of n nodes requires at least $\lfloor log_2 n \rfloor + 1$ bits to represent the asserted node at that level. The size of a population member is $(\lfloor log_2 n \rfloor + 1)(m+2)$ bits under this scheme, which is a large reduction in the search space.

The representation still suffers, because the symbols possess no "semantic" information. For example, in [92], Gray coding is shown to be superior to binary coding because the Gray representation explicitly encodes the concept of domain continuity into each GA member. In other words, members that are near in a Hamming distance sense perform similarly on the cost function in question. Thus, the Gray coding adds semantic information to the representation of the search point.

This same idea of coding semantic information into each GA member is at the crux of the Principle of Meaningful Building Blocks. Similar short, low-order schemata should possess similar meanings when mapped from the function domain onto the binary representation. In this case, the function domain is the set of symbols in the ARM.

Antoinesse and Keller [86] examine coding higher level representations (similar to ARM symbols) into binary strings. In their representation of a symbol, each bit contains semantic information about the symbol, such as the class of objects that the symbol belongs to, or features of the object that the symbol represents. In this way, similar objects in the real world possess similar representations in the

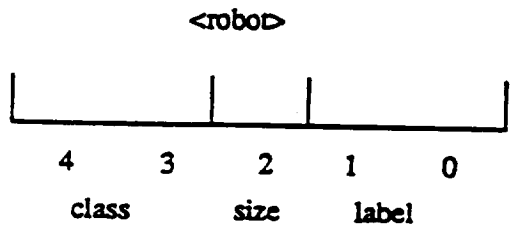
GA domain. Again, if similar symbols possess similar representations, the GA can effectively search for "features" that perform well on the target function, by promoting schemata that have these features present. It is important to note that for the study in [86], as well as for the ARM, the symbol representation is context-dependent, i.e., depending on the objects in the current world, the representation changes.

For the ARM system, a similar strategy is adopted. Each bit, or group of bits in the representation of a symbol encodes some real-world semantic information about the *ACTOR*, and *OBJ*s symbols. Semantic information that may be encoded includes:

1. Major class
2. Minor class
3. Actor significant features (weak vs. strong, dextrous vs. clumsy, human vs. machine)
4. Object significant features (heavy vs. light, big vs. small, active vs. inert)

It is a little more difficult to determine similar features between actions. However, actions that produce the same *STATE* symbol in the rule effect should have encodings that are near in a Hamming distance sense.

An example of the codings for a set of *ACTOR* nodes is presented in Figure 4.30. Notice that the first bits of the code are devoted to semantic information, while the last bits contain labels to distinguish between semantically similar actors. This example demonstrates that similar symbols have meaningful building blocks, and are nearby, in a Hamming distance sense.



BITS:

4-3: 00 = <dex>, 01 = <pos>, 10 = <trans>
2: 0 = light, 1 = heavy

ENCODING:

00000: EVA	00001: JRMS	00010: FTS	00011: SPDM
00100: GDMS	00101: MRMS		
01000: SRMS	01001: SSRMS		
01100: ATD	01101: APS		
10000: MMU	10001: CETA	10010: MT	
10100: OMV	10101: ORBITER		

Figure 4.30: Encoding of Actor nodes for the GA

4.6 Finding Sets of High POE Robotic Actions

The Genetic Algorithm possess the capability of finding a high POE robotic action given a desired effect. It may be the case, however, that the found robotic action cannot be used due to other planning constraints, such as resource utilization. It would be very helpful, therefore, if the GA could produce a *set* of robotic actions, each of which possesses a high POE value for the given effect. If one robotic action could not be used, another would be selected from this set and applied to the planning problem.

Finding sets of high POE robotic actions requires the GA to find multiple minima in the energy function of the ARM. Since the GA tends to converge to a uniform population as the search progress, this requirement seems contradictory to the nature of the GA. Several researchers, however, have examined the problem of multiple solutions, or *speciation*, using a GA.

To increase the diversity of a GA population and allow for speciation, DeJong [72] introduces a crowding scheme to the GA. In this scheme, existing members are replaced in the population based on their similarity with other population members. This replacement occurs when children are generated through crossover. It is shown that as the search progressed, stable species, or function optima, are found using this technique.

Goldberg et al. [94, 95] develop a sharing function to promote speciation and find multiple optima. This research uses the distance between the mapped GA members to determine the amount of fitness like members should "share." The technique follows:

Let

$$d_{ij} = d(\mathbf{X_i}, \mathbf{X_j}) \tag{4.17}$$

where d is a distance measure and $\mathbf{X_i}$ and $\mathbf{X_j}$ are two population members. We define a sharing function sh with the following three properties.

1. $0 \leq sh(d_{ij}) \leq 1$ for all d_{ij}.

2. $sh(0) = 1$.

3. $\lim_{d_{ij} \to \infty} sh(d_{ij}) = 0$.

Many sharing functions can be used, for example

$$sh(d_{ij}) = \begin{cases} 1 - (\frac{d_{ij}}{\sigma_{share}})^{\eta}, & d_{ij} < \sigma_{share} \\ 0, & otherwise \end{cases} \tag{4.18}$$

where σ_{share} and η are positive constants. Given a distance metric, a sharing function, and the fitness of a member i by f_i, the shared fitness, f_i' of member i is given by:

$$f_i' = \frac{f_i}{m_i}, \text{ if } m_i > 0 \tag{4.19}$$

where

$$m_i = \sum_j sh(d_{ij}) \tag{4.20}$$

In effect, this algorithm reduces the fitness of each member which is similar to other members in the population. This allows diversification, because new members can compete for selection. The research demonstrated the ability to find multiple peaks in various trigonometric functions.

For the associative recall in the ARM, a technique similar to Goldberg's sharing function is used to find multiple robotic actions. Since neighboring solutions in the ARM may each represent high POE robotic actions, only identical members should be penalized by a reduction in fitness. The penalty method we use is as follows.

1. Create sets of identical members.

2. Enumerate the members of each set j by assigning each member an index i, $0 \leq i < Setsize(j)$.

3. For each set j do

(a) Since all members in set j are identical, they possess identical fitness values. Let f_j be the fitness of each member in set j.

(b) For each member i in set j, assign i a new fitness, $f_i' = \mu^i f_j$.

If $\mu < 1$, the fitnesses of identical members decay exponentially. In other words, the first member of a set of identical members in the population will have its assigned fitness value, f_i. The second member of the identical set will have μf_i as a fitness value. The third member will have $\mu^2 f_i$ as a fitness value, etc. Of course, unique members will not have their fitness values altered by this method. This technique allows diversity and speciation, since it inhibits a uniform population from occurring.

The case study in Chapter 6 demonstrates the effectiveness of this method at finding multiple robotic actions for a given effect.

4.7 Contributions and Conclusions

This chapter detailed the associative recall technique for the ARM. The main contributions of this chapter are as follows.

1. The development of the immigration operator for Genetic Algorithms and the demonstration that immigration improves the performance of a GA on functions that possess difficult local optima.

2. The proof that a GA combined with the immigration operator will converge in probability to the global optimum of a cost function.

The nature of the energy function of the ARM was described, and was shown to be highly nonlinear and discrete. Based on this information, the Genetic Algorithm and Simulated Annealing were tested as possible optimization functions to perform associative recall on the ARM. Experimental results in this chapter showed that the GA can perform as least as well as Simulated Annealing when searching for the

minimum energy of a Boltzmann Machine. Based on this study, the GA was chosen as the associative recall technique for the ARM.

The immigration operator was introduced and examined in terms of the tradeoff between exploration and exploitation in a Genetic Algorithm. A test suite of functions were defined to examine this tradeoff. The functions were characterized by difficult local optima. Two types of GAs were tested with and without immigration on the test suite Steady State and Restarted. It was shown that a GA with immigration reduces the number of function evaluations required to find the global minimum of a cost function, and also reduces the number of outliers. Further research using a generational GA was recommended.

The concept of convergence was discussed. Emphasis was placed on the difference between bitwise convergence of the population and convergence to a global optimum. The GA with immigration was shown to converge in probability to the global optimum of a cost function.

The issue of representation of ARM symbols in a GA member was discussed. It was decided to encode semantic information in the symbols to allow the GA to exploit the underlying structure of the members. Finally, a method for finding sets of high POE robotic actions for a given effect was outlined. The technique chosen is similar to those in the GA literature and is based on the concept of speciation.

CHAPTER 5
A BOLTZMANN MACHINE FOR THE ORGANIZATION OF INTELLIGENT MACHINES

The purpose of this chapter is to relate the design of the ARM to the concept of Intelligent Machines. In particular, the ARM can form the Organization level of the Intelligent Machine, as defined by Saridis. Much of this chapter has appeared in [16].

Since 1977 Saridis has been developing a novel approach, defined as Hierarchical Intelligent Control, designed to organize, coordinate and execute anthropomorphic tasks by a machine with minimum interaction with a human operator. This approach utilizes analytic (probabilistic) models to describe and control the various functions of the Intelligent Machine structured by the intuitively defined principle of Increasing Precision with Decreasing intelligence (IPDI) as presented in [96].

This principle, which resembles the managerial structure of organizational systems [97], has been derived on an analytic basis by Saridis [98]. The impact of this work is in the *engineering* design of intelligent robots, since it provides analytic techniques for universal production (blueprints) of such machines.

The outline of the chapter follows. In section 5.1 some mathematical theory of the Intelligent Machine is outlined. In section 5.2, some definitions of the variables associated with the principle, such as Machine Intelligence, Machine Knowledge, and Precision are made [18]. Section 5.3 describes the procedure to establish a Boltzmann machine, such as the ARM, on an analytic basis as the Organization level. Section 5.4 concludes this chapter.

5.1 The Mathematical Theory of Intelligent Controls

To design intelligent machines that require their operation control system to possess intelligent functions such as simultaneous utilization of a memory, learning, or multilevel decision making in response to "fuzzy" or qualitative commands, The theory of Intelligent Controls has been developed by Saridis [99, 13]. It utilizes the results of cognitive systems research effectively with various mathematical programming control techniques [100].

The theory of Intelligent Control systems, proposed by Saridis [12] combines high level decision making with advanced mathematical modeling and synthesis techniques of system theory with linguistic methods of dealing with imprecise or incomplete information. This produces a unified approach suitable for the engineering needs of the future. The theory may be thought of as the result of the intersection of the three major disciplines of Artificial Intelligence, Operations Research, and Control Theory. This research is aimed to establish Intelligent Controls as an engineering discipline, and it plays a central role in the design of Intelligent Autonomous Systems.

The control intelligence is hierarchically distributed according to the *Principle of Increasing Precision with Decreasing Intelligence* (IPDI), evident in all hierarchical management systems. They are composed of three basic levels of controls even though each level may contain more than one layer of tree-structured functions as shown in Figure 5.1. These levels are:

1. The Organization level.
2. The Coordination level.
3. The Execution level.

The Organization level, as shown in Figure 5.2, is intended to perform such operations as planning and high level decision making from long term memories. It may require high level information processing such as the knowledge based systems

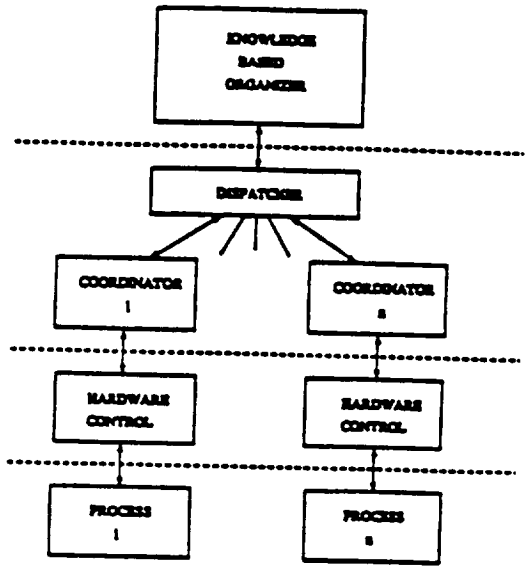


Figure 5.1: Intelligent Machine Hierarchy

encountered in Artificial Intelligence. These require large quantities of knowledge processing but require little or no precision.

The functions involved in the upper levels of an Intelligent Machine are imitating functions of human behavior and may be treated as elements of knowledge-based systems. Actually, the activities of planning, decision making, learning, data storage and retrieval, task coordination, etc. may be thought of as knowledge handling and management. Therefore, the flow of knowledge in an Intelligent Machine may be considered as the key variable of such a system.

Knowledge flow in an Intelligent Machine's organization level is present in the following high level activities.

1. Data Handling and Management.
2. Planning and Decision Making performed by the central processing units.
3. Sensing and Data Acquisition obtained through peripheral devices.
4. Formal Languages which define the software.

The uncertainty present in the knowledge of the Organization level can be

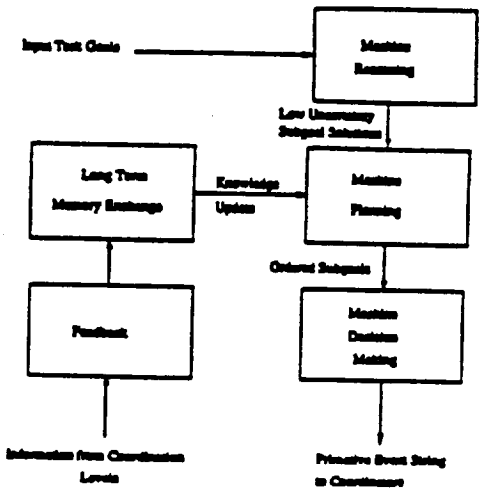


Figure 5.2: Organization Level of Intelligent Machine

represented by an analytic function and serves as a measure of the performance of this level at different activities. Saridis has chosen the *entropy* function as an analytic measure to represent the uncertainty [12] of the Organization level.

The Coordination level is an intermediate structure serving as an interface between the organization and execution level. Its performance is also measured by an entropy function.

The Coordination level is involved with coordination of actions, decision making and learning on a short term memory, e.g., a buffer. As input, the Coordination level receives high level planning steps from the Organization level. Each step in the plan dictates an action that must be performed in the environment of the Intelligent Machine. The Coordination level maps the high level plan onto the environment by selecting detailed actions that have a minimum entropy of execution. These entropies are derived from subjective probabilities provided initially and updated by feedback from the Execution level. The detailed actions are then provided to the Execution level for execution in the environment.

Recent work in the Coordination level includes communication protocols formulated by Wang [101, 102] and a collision avoidance coordinator for mobile robots, developed by Kyriakopoulos [103].

The Execution level executes the appropriate actions in the environment, which are represented as control functions at this level in the Intelligent Machine hierarchy. Its performance measure can also be expressed as an entropy, thus unifying the performance measures of an Intelligent Machine.

Optimal control theory utilizes a non-negative functional of the states of a system in the state space, and a specific control from the set of all admissible controls,to define the performance measure for some initial conditions $(x(t), t)$, representing a generalized energy function. Minimization of the energy functional yields the desired control law for the system.

For an appropriate density function $p(x, u(x,t), t)$ satisfying Jaynes' Maximum entropy principle [104], it was shown by Saridis that the entropy $H(u)$ for a particular control action $u(x,t)$ is equivalent to the expected energy or cost functional of the system [98]. Therefore, minimization of the entropy $H(u)$ yields the optimal control law of the systems. Therefore, since the actions executed by the Execution level are represented as control functions, their performance can be evaluated by an entropy measure.

Since all levels of a hierarchical intelligent control system can be measured by entropies and their rates, the optimal operation of an "intelligent machine" can be obtained through the solution of mathematical programming problems. Since entropy satisfies the additive property, this programming problem must minimize the total entropy of activities in the Intelligent Machine.

The various aspects of the theory of hierarchically intelligent controls are summarized by Saridis as follows [15].

> The theory of intelligent machines may be postulated as the mathematical problem of finding the right sequence of decisions and controls for a system structured according to the principle of increasing precision with decreasing intelligence (constraint) such that it minimizes its total entropy.

The above analytic formulation of the "Intelligent Machine problem" as a hierarchically intelligent control problem is based on the use of entropy as a measure of performance at all the levels of the hierarchy. It has many advantages because of the tree–like structure of the decision making process, and brings together functions that belong to a variety of disciplines.

5.2 Knowledge Flow and the Principle of IPDI

The general concepts of Intelligent Control Systems are the fundamental notions of Machine Intelligence, Machine Knowledge, its Rate and Precision. The following definitions are made by Saridis to elucidate these concepts.

Definition 5.1. Machine Knowledge is defined to be the structured information acquired and applied to remove ignorance or uncertainty about a specific task pertaining to the Intelligent Machine.

Definition 5.2. Rate of Machine Knowledge is the flow of knowledge through an Intelligent Machine.

Intelligence is defined by the American Heritage Dictionary of the English Language [105] as: *Intelligence* is the capacity to acquire and apply knowledge. In terms of Machine Intelligence, this definition is modified to yield:

Definition 5.3. Machine Intelligence (MI) is the set of actions which operates on a database (DB) of events to produce flow of knowledge (R).

One may directly apply the Law of Partition of Information Rates of Conant [106] to analyze the functions of the intelligence within the activities of an Intelligent Control System.

Definition 5.4. Imprecision is the uncertainty of execution of the various tasks of the Intelligent Machine.

On the other hand, one may define Precision as follows:

Definition 5.5. Precision is the complement of Imprecision, and represents the complexity of a process.

Analytically, Saridis summarizes the relationships as follows.

Knowledge (K) representing a type of information may be represented as

$$K = -\alpha - ln\{P(K)\} \equiv (Energy) \tag{5.1}$$

where $P(K)$ is the probability density of knowledge and α is a probability normalizing constant.

From (5.1) the probability density function $P(K)$ satisfies the following expression in agreement with Jaynes' principle of Maximum Entropy [104]:

$$P(K) = e^{-\alpha - K}; \quad \alpha = ln \int_X e^{-K} dx \tag{5.2}$$

The Rate of Knowledge R which is the main variable of an Intelligent Machine with discrete states is

$$R = \frac{dK}{dt} = (Power) \tag{5.3}$$

where t represents time.

It was intuitively thought by Saridis that the Rate of Knowledge must satisfy the following relation which may be thought of expressing the principle of Increasing Precision with Decreasing Intelligence [13].

$$(MI) : (DB) \longrightarrow (R) \tag{5.4}$$

A special case is when R is fixed, machine intelligence is largest for a smaller data base. This is in agreement with Vamos' theory of Metalanguages [107]. It is interesting to notice the resemblance of this entropy formulation of the Intelligent Control Problem with the ϵ–entropy formulation of the metric theory of complexity originated by Kolomogorov [108] and applied to system theory by Zames [109]. Both methods imply that an increase in knowledge (feedback) reduces the amount of entropy (ϵ–entropy) which measures the uncertainty involved with the system.

Therefore, the analytic formulation of the above principle has been derived by Saridis from simple probabilistic relations among the Rate of Knowledge, Machine Intelligence and the Database of Knowledge. The entropies of the various functions come naturally into the picture as a measure of their activities.

5.3 The Organization Level as a Boltzmann Machine.

In the current literature of parallel architectures for Machine Intelligence, the Boltzmann Machine represents a powerful. neural network based architecture that allows efficient searches to optimally obtain the combination of certain input variables and constraints [55]. The formulation of the ARM demonstrates this capability.

The Boltzmann architecture may be interpreted as the machine that searches for the optimal interconnection of several nodes (neurons) representing different

primitive events to produce a string defining an optimal task. Such a device may prove extremely useful for the design of the Organization level of an Intelligent Machine [18].

One of the main functions of the Organization level is to construct a set of primitive events which represent an activity to be executed by the Intelligent Machine in order to achieve a desired goal. Primitive events consist of actors, actions and objects which combine to form the robotic action. The set of primitive events which form the activity must also minimize the uncertainty of achieving the goal. Mapping these requirements onto a Boltzmann Machine, we can define the following sets of nodes:

1. Input nodes that represent a desired goal or subgoal.

2. Output nodes that represent primitive events which must be executed by the Intelligent Machine to satisfy the input goal.

3. Hidden nodes which allow the development of complex interactions between input and output nodes.

As we can see, this is structurally identical to the model of the ARM.

We associate the state of each node with a binary random variable $n_i = \{0, 1\}$, with a priori probabilities $p(n_i = 1) = p_i$. $p(n_i = 0) = 1 - p_i$, where 1 represents the assertion of node i, and 0 indicates node i is not asserted. For the standard definition of the Boltzmann Machine, and the ARM model $p_i = 0.5$. The state vector of the network, $\mathrm{N} = (n_1, n_2, \ldots, n_i, \cdots, n_m)$ is an ordered set of 0's and 1's describing the state of the machine in terms of its nodes, for an m node machine. It is possible to extract the string of primitive events representing the optimal task by examining the state vector of the output nodes in the network in steady state response to a given input.

The standard formulation of the Boltzmann Machine uses energy as a cost function which is minimized to find the optimal state of the machine. However in (5.1), knowledge is defined as a form of energy. What energy represents, therefore, is the ignorance possessed in the knowledge about a particular machine state N. As the energy increases, the ignorance increases as well. Further, one can analytically represent the probability that a correct set of primitive events has been found for a particular input goal based on the knowledge in the machine about the input-output pair. Finally, the uncertainty of the input-output pattern N can be computed as an entropy function.

5.4 Entropy as a Measure of Uncertainty

Entropy is used as a measure of uncertainty in the Intelligent Machine. The entropy manifests itself in the interaction and interconnection of nodes in the network. Let us begin by defining the ignorance of knowledge about a particular machine state N by the energy function

$$K(\mathrm{N}) = \frac{1}{2}\sum_{i}\sum_{j} w_{ij}n_j n_i \tag{5.5}$$

This is identical to the energy function of the ARM, as given in (3.10).

The probability that the output primitive event nodes are correct given the input is a function of the ignorance in the knowledge about state N and is given by

$$P(K(\mathrm{N})) = exp(-\alpha - \frac{1}{2}\sum_{i}\sum_{j} w_{ij}n_j n_i) \tag{5.6}$$

where

- w_{ij} is the interconnection weight between nodes i and j
- $w_{ii} = 0$
- α is a probability normalizing factor.

We now wish to formulate a measure of uncertainty for the machine. A standard entropy formulation is given by:

$$H(\mathbf{X}) = -\sum_{\mathbf{X}} P(\mathbf{X}) ln\{P(\mathbf{X})\} \tag{5.7}$$

Let us adapt this measure to reflect the uncertainty that the machine produces as output the correct set of primitive events that achieve a desired goal. The adapted measure has two states,

1. a correct set of primative events,

2. an incorrect set of primative events.

This yields a two state entropy measure for machine state N, where $P(K(\mathbf{N}))$ is the probability that the output is correct (state 1), and $1 - P(K(\mathbf{N}))$ is the probability that the output is incorrect (state 2).

The uncertainty that the output of the Boltzmann Machine is correct is given by:

$$H(K(\mathbf{N})) = -P(K(\mathbf{N}))ln\{P(K(\mathbf{N}))\} - (1 - P(K(\mathbf{N})))ln\{(1 - P(K(\mathbf{N})))\} \tag{5.8}$$

The entropy is maximum when the each of the associated probabilities $P(K) = \frac{1}{2}$. Maximum entropy implies complete uncertainty in a decision and reflects lack of preference on a correct string configuration. By bounding $P(K)$ from below by $\frac{1}{2}$, one obtains a unique minimization of the entropy corresponding to the most certain sequence of events possible which achieve a given goal.

Further, by using the formulation provided above for entropy, one can minimize the uncertainty of the Organization level in producing a string of primitive events which achieve a desired task by finding a configuration of node states which minimize the energy in the Boltzmann Machine. Therefore, using this measure, one can find a minimum entropy output by minimizing the energy in the machine.

5.5 Contributions and Conclusions

The main contribution of this chapter was the reformulation of the Organization level of the Intelligent Machine as a Boltzmann Machine, and to demonstrate that the ARM could function in this capacity. The chapter also provided some background on Intelligent Machines, and presented the fundamental definitions. It was shown that the knowledge of the Organization level could be represented by an energy function. Also, finding the set of primitive events which minimizes the entropy for a given input was shown to correspond to to a minimum energy search of the network.

Equivalence between the Organization level and the ARM allows the former to use the ARM training procedure to develop connection weights. This indicates that the Organization level can be used to predict entropy values for untested combinations of primitive events. Also, any optimization technique used for associative recall in the ARM can be used to search the Organization level and find an activity, or set of activities which minimizes the entropy for a given input goal.

CHAPTER 6
A CASE STUDY

To demonstrate the capabilities of the Associative Rule Memory, this chapter presents a case study that employs the ARM for a typical robotic application. This case study is based upon the *Task Analysis Methodology* (TAM) for the NASA Flight Telerobotic Servicer [2]. Section 6.1 of this chapter broadly describes the TAM. In section 6.2, we outline the case study plan and provide measures to evaluate the performance of the ARM. Section 6.3 begins development of the ARM for the case study by presenting the target world model, the general and specific rules, and the resulting network model. Section 6.4 details the training sets that form the experiments for the case study. Also, the results of training are presented and analyzed. Section 6.5 describes the results of associative recall, and the optimal planning steps. Section 6.6 reviews the results, and concludes this chapter.

6.1 The Task Analysis Methodology

As described in [2], the TAM:

- Provides a method to develop operational scenarios for telerobotic systems.
- Provides a method to analyze and evaluate telerobotic systems task performance capabilities.
- Provides a common language for space station telerobotic users (i.e., operational planners, hardware and software developers, and program managers).
- Provides a method to optimize telerobotic operations on the Space Station Freedom by assessing task scenarios and recommending task and hardware design requirements.

- Provides a standard format for inputting operational scenarios to off-line planning software.

To provide these capabilities, the TAM details:

- A set of tasks (actions) that can be accomplished by work systems (agents) in the Space Station environment. These tasks are arranged hierarchically with increasing levels of detail.
- A set of work systems (agents) and their capabilities.
- A set of objects in the world model on which actions are performed.

Each action in the TAM can be divided into a set of more detailed sub-actions to create plans with different levels of abstraction. Similarly, the set of effects of robotic actions on the world model can be represented in several levels of detail. These tiers of abstraction are called the *Task Analysis Hierarchy* (TAH). The TAM planning process, described below, is repeated for each level of detail in the TAH.

A flowchart detailing the TAM planning process is presented in Figure 6.1. The planning process is described using the terminology we have developed in this thesis. The description follows.

1. *Task Statement.* A world model goal is provided as input to the planning process.
2. *Task Decomposition.* The world model goal is decomposed into a more detailed set of effects that must take place in the world to achieve that goal.
3. *Task Scripting.* Using available agents, a set of robotic actions are defined that can accomplish each of the effects.

4. *Task Modeling and Assessment.* Upon completion of a suitable plan, robotic actions are modeled using a mockup of the world and performance is assessed. Constraints and modifications are outlined, if necessary, and replanning occurs.

Examining the capabilities and features of the TAM planning process, we see that it is very similar to an interactive planning system, but also embodies features of automatic planners, such as Rokey's TIPS planner [11]. For example, step 2 of the TAM planning process is similar to level 1 of the TIPS planner, while step 3 is contained within level 2 of the TIPS planner.

A feature of the Space Station world model, as described by the TAM document, is the redundancy of robotic actions. In other words, a particular desired effect, provided by the Task Decomposition process, can be achieved by several (perhaps many) different robotic actions. To limit the number of possible robotic action alternatives for a given desired effect, the Task Scripting process must efficiently determine a set of robotic actions that achieve the desired effect and optimize some performance criteria. As we can see, this requires the same functionality provided by the ARM model where the performance criteria is the POE value of a robotic action for a desired effect. This case study, therefore, models step 3 of the TAM planning process, Task Scripting, that requires some of the capabilities that are provided by the ARM.

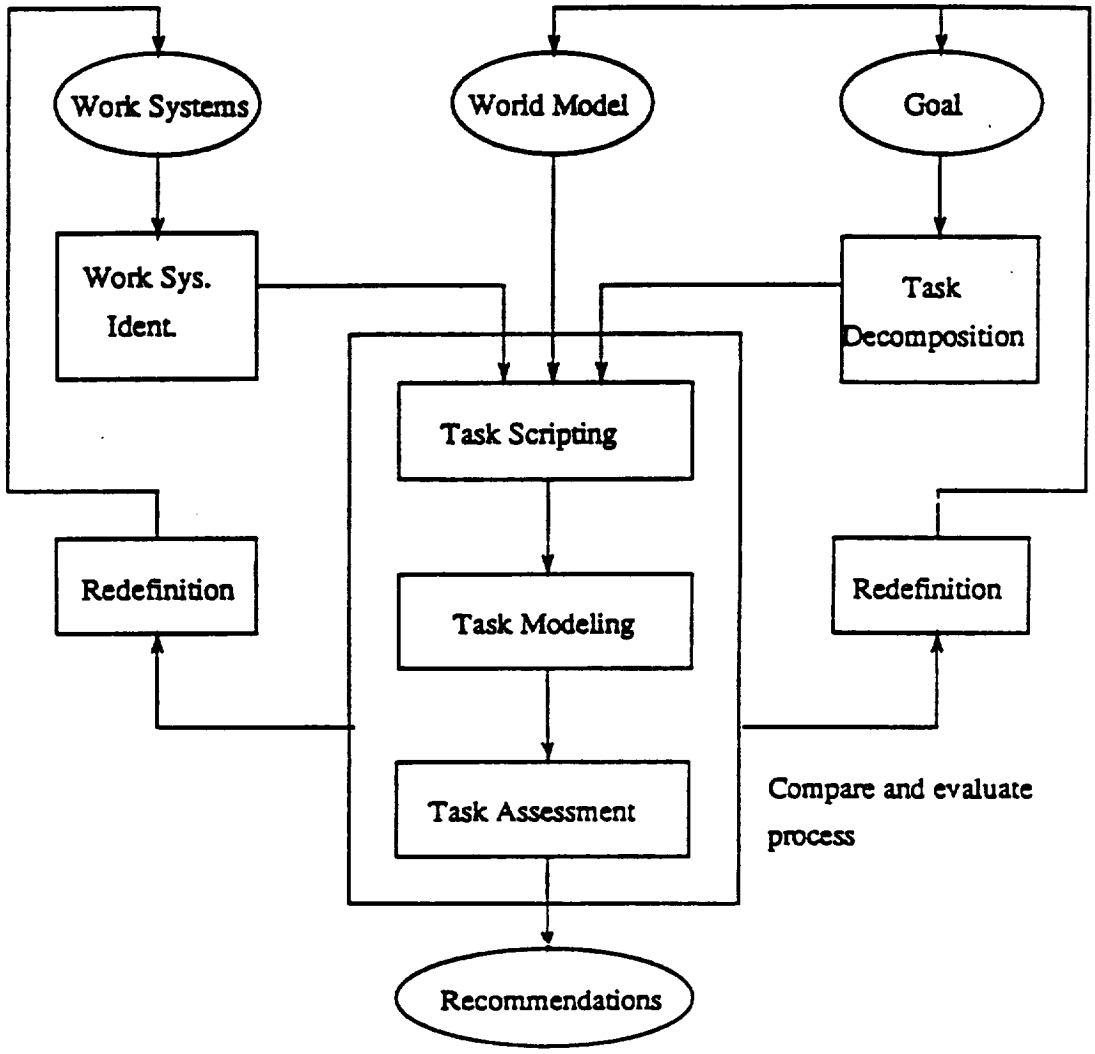


Figure 6.1: Task Analysis Methodology Flowchart

6.2 Case Study Goal

To test the functionality of the ARM model, it is applied to a typical Space Station task as described in the TAM documentation. The task named "ORU CHANGEOUT" has been adapted for this case study. In this scenario, the Orbital Replacement Unit, which is attached to the Truss structure, must be deactivated, moved to the Mobile Servicing Center and stowed. The steps follow.

Goal: ORU CHANGEOUT

1. The ORU is deactivated.
2. The ORU is detached from the Truss.
3. The ORU is attached to a carrier.
4. The carrier is moved to the Mobile Servicing Center.
5. The ORU is detached from the carrier.
6. The ORU is attached to the Mobile Servicing Center.
7. The ORU is stowed at the Mobile Servicing Center.

It is the responsibility of the ARM to find high POE robotic actions that achieve each of the steps in the above plan. This plan will be called the *goal plan* for the rest of this chapter.

Using this sample plan, the ARM will be evaluated with several criteria:

1. The ability to represent general rules.
2. The ability to accurately store specific rules and corresponding POE values from the training set.
3. The ability to predict POE values for untested specific rules.

4. The speed of associative recall of the optimal robotic action for each desired effect of the plan.

5. The speed of associative recall of a set of near-optimal robotic actions for each desired effect of the plan.

6.3 Design of the Case Study using the Task Analysis Methodology

To determine a set of robotic actions that accomplish the goal plan, we must first develop the ARM model. This is accomplished as follows:

1. Determine the target world model.

2. Define the symbol classes for agents in the world model.

3. Develop a set of general rules.

4. Create a suitable ARM network from the symbols, symbol classes and general rules.

6.3.1 The world model and symbol classes

This study focuses on the Telerobotic level of the TAH. This level possesses the required degree of abstraction within which the ARM is suited to function. From this level, a world model is defined consisting of actors and objects. The actors are telerobots and consist of

- Dextrous Manipulators
- Positioners
- Transporters

The objects in the world consist of

- Telerobots
- Tools
- Carriers and Platforms
- Sites
- Other parts

A full breakdown of the symbol classes is presented in Appendix B along with the set of symbols that fit in each class. The agents used in this study are a combination of agents from the TAM document, and agents defined specifically for this study. The extra agents are added to show the full capabilities of the ARM by increasing the redundancy of the environment. The classification is accomplished heuristically by grouping agents together that are similar. Other classifications may also be possible and valid.

6.3.2 A general rule grammar

The general rule grammar is designed to represent a high degree of robotic action functionality. It includes the following capabilities:

1. The ability for two robots to work together to accomplish a task.
2. The ability for a tool to be used in accomplishing a task.

The general rule grammar is defined as

$$ACTOR\ ACTION\ OBJ_D\ OBJ_I\ SLAVE\ TOOL\ \rightarrow$$
$$OBJ_D\ STATE\ OBJ_I \tag{6.1}$$

where

- $ACTOR$ is limited to class $< robot >$.

- $ACTION$ is an action symbol.
- OBJ_D is the direct object class and is limited to $< object >$.
- OBJ_I is the indirect object class and is limited to $< object >$.
- $SLAVE$ is a secondary robot and is limited to class $< robot >$.
- $TOOL$ is a limited to class $< tools >$.
- $STATE$ is a state symbol.

Using this structure, the general rules generated from the TAM documentation are presented in Figures 6.2 and 6.3.

Following the conventions outlined in Chapter 3, the direct object of the robotic action must be the same as the direct object of the desired effect. Similarly, the indirect object of the robotic action must be the same symbol as the indirect object in the desired effect.

The general rules provide a set of actions, many of which lead to the same state in the rule effect. Different action symbols that lead to the same effect represent real-world, alternative methods for achieving the state. As many of these rules show, different methods often require different sets of $SLAVE$s or $TOOL$s to achieve the desired effect.

Some of the actions also lead to multiple effects, such as $HOLD$ and $RETRACT$. This is to demonstrate that the ARM is capable of representing multiple effects of general rules.

Also, the agent symbols include a $BLANK$ value, which can be used when the robotic action does not require an OBJ symbol to be successful. For example, although the action $STOW$ allows a $< fix >$ symbol, a fixturing tool may be unnecessary to achieve a high POE value. If this is the case, the $< fix >$ variable

- < *dex* > *ACTUATE* < *object* > *NULL NULL* < *active* > →
  < *object* > *IS – ACTIVATED NULL*

- < *dex* > *ATTACH* < *object* > < *object* > < *dexpos* > *NULL* →
  < *object* > *IS – ATTACHED – TO* < *object* >

- < *dex* > *CONNECT* < *object* > < *object* > *NULL* < *fix* > →
  < *object* > *IS – ATTACHED – TO* < *object* >

- < *dex* > *DEACTIVATE* < *object* > *NULL* < *dexpos* > < *active* >
  →
  < *object* > *IS – DEACTIVATED NULL*

- < *dex* > *DEACTUATE* < *object* > *NULL NULL* < *active* > →
  < *object* > *IS – DEACTIVATED NULL*

- < *dex* > *DEPLOY* < *object* > *NULL* < *dex* > < *active* > →
  < *object* > *IS – ACTIVATED NULL.*

- < *dex* > *DEPLOY* < *object* > < *object* > < *dex* > < *active* > →
  < *object* > *IS – AT* < *object* >.

- < *dex* > *DETACH* < *object* > < *object* > < *pos* > *NULL* →
  < *object* > *IS – DETACHED – FROM* < *object* >

- < *dex* > *DISCONNECT* < *object* > < *object* > *NULL* < *defix* >
  →
  < *object* > *IS – DETACHED – FROM* < *object* >

- < *dex* > *FASTEN* < *object* > < *object* > < *dexpos* > < *fix* > →
  < *object* > *IS – ATTACHED – TO* < *object* >

- < *dex* > *HOLD* < *object* > < *object* > < *dexpos* > < *fix* > →
  < *object* > *IS – AT* < *object* >.

- < *dex* > *HOLD* < *object* > < *object* > < *dexpos* > < *fix* > →
  < *object* > *IS – ATTACHED – TO* < *object* >.

Figure 6.2: Case Study General Rules

- $< dex >\ OPERATE\ < object >\ NULL\ < dexpos >\ < active > \rightarrow$
  $< object >\ IS-ACTIVATED\ NULL.$
- $< dexpos >\ PLACE\ < object >\ < object >\ < dexpos >\ NULL \rightarrow$
  $< object >\ IS-AT\ < object >.$
- $< robot >\ POSITION\ < object >\ < object >\ NULL\ NULL \rightarrow$
  $< object >\ IS-AT\ < object >.$
- $< robot >\ RETRACT\ < object >\ NULL\ < dexpos >\ < active > \rightarrow$
  $< object >\ IS-DEACTIVATED\ NULL$
- $< robot >\ RETRACT\ < object >\ < object >\ < dexpos >\ < active >$
  $\rightarrow$
  $< object >\ IS-AT\ < object >$
- $< robot >\ STOW\ < object >\ < object >\ < dex >\ < fix > \rightarrow$
  $< object >\ IS-STOWED-AT\ < object >$
- $< trans >\ TRANSPORT\ < object >\ < object >\ NULL\ NULL \rightarrow$
  $< object >\ IS-AT\ < object >$
- $< dex >\ UNFASTEN\ < object >\ < object >\ < dexpos >\ < defix >$
  $\rightarrow$
  $< object >\ IS-DETACHED-FROM\ < object >$
- $< robot >\ UNSTOW\ < object >\ < object >\ < dex >\ < defix > \rightarrow$
  $< object >\ IS-UNSTOWED-TO\ < object >$

Figure 6.3: Case Study General Rules, cont'd.

would be instantiated with the *BLANK* symbol during recall, to develop a high POE robotic action.

6.3.3 The ARM network

Given the structure of the general rules, the ARM network is comprised of three input levels and six output levels. The input levels are

1. OBJ_D
2. $STATE$
3. OBJ_I

The output levels are

1. $ACTOR$
2. $ACTION$
3. OBJ_D
4. OBJ_I
5. $SLAVE$
6. $TOOL$

From the discussion in Chapter 3, one node is used to represent each symbol on each level. The input level nodes are presented in Figure 6.4. Similarly, the set of nodes that form the output levels are presented in Figure 6.5. The set of all input and output node combinations allows us to represent the possible robotic actions required to achieve the goal plan.

1. **OBJ_D**: *FTS, SPDM, JRMS, EVA, GDMS, MRMS, SRMS, SSRMS, ATD, APS, MMU, MT, CETA, OMV, ORBITER, GLUEGUN, WELDER, BOLTER, PINS-H, PINS-M, PINS-L, CLAMP-H, CLAMP-M, CLAMP-L, GRAPPLER-H, GRAPPLER-M, GRAPPLER-L, PRYBAR, SEPARATOR, DEMATOR, TOOLSET0, TOOLSET1, TOOLSET2, TOOLSET3, TOOLSET4, TOOLSET5, TOOLSET6, CARRIER-L, CARRIER-M, CARRIER-S, PALLET-L, PALLET-M, PALLET-S, ORU, TRUSS, AWP, MSC*

2. **STATE**: *IS-ACTIVATED, IS-AT, IS-ATTACHED-TO, IS-DEACTIVATED, IS-DETACHED-FROM, IS-INSIDE-OF, IS-STOWED-AT, IS-UNSTOWED-TO*

3. **OBJ_I**: *BLANK, CARGO-BAY, AIRLOCK, FTS, SPDM, JRMS, EVA, GDMS, MRMS, SRMS, SSRMS, ATD, APS, MMU, MT, CETA, OMV, ORBITER, CARRIER-L, CARRIER-M, CARRIER-S, PALLET-L, PALLET-M, PALLET-S, ORU, TRUSS, AWP, MSC*

Figure 6.4: Input levels and nodes for case study network

1. **ACTOR:** *FTS, SPDM, JRMS, EVA, GDMS, MRMS, SRMS, SSRMS, ATD, APS, MMU, MT, CETA, OMV, ORBITER*

2. **ACTION:** *ACTUATE, ATTACH, CONNECT, DEACTIVATE, DEACTUATE, DEPLOY, DETACH, DISCONNECT, FASTEN, HOLD, OPERATE, PLACE, POSITION, RETRACT, STOW, TRANSPORT, UNFASTEN, UNSTOW*

3. **OBJ_D**: This list is the same as the OBJ_D list for the input level.

4. **OBJ_I**: This list is the same as the OBJ_I list for the input level.

5. **SLAVE**: This list is the same as the *ACTOR* level for the output level, but also includes a *BLANK* node.

6. **TOOL:** *GLUEGUN, WELDER, BOLTER, PINS-H, PINS-M, PINS-L, CLAMP-H, CLAMP-M, CLAMP-L, GRAPPLER-H, GRAPPLER-M, GRAPPLER-L, PRYBAR, SEPARATOR, DEMATOR, TOOLSET0, TOOLSET1, TOOLSET2, TOOLSET3, TOOLSET4, TOOLSET5, TOOLSET6, BLANK*

Figure 6.5: Output levels and nodes for case study network

6.4 Case Study Experiments

The case study experiments are constructed to simulate tests performed in a real-world, Space Station telerobotic environment.

6.4.1 Experimental Procedure

The test suite is comprised of seven training/knowledge sets, which correspond to the seven steps in the goal plan presented in section 6.2. Initially, the network is trained on each set individually to demonstrate that the network can accurately represent the specific rules and POE values in each of the training sets. General rules and knowledge rules are then added to the network. Predictive capabilities of the ARM network are examined for each training set. The network is reset after each test, and the next training set is evaluated.

The network is then trained on all the training sets combined. Accuracy of training is examined for this large test case. The general rules and all the knowledge rules are added to the network. Predictive capabilities are tested, and it is demonstrated that combining the training sets creates symbolic relationships that were not present in each of the individual training sets.

6.4.2 Experimental Suite

The seven training/knowledge sets are presented in Figures 6.6 - 6.12. The training sets are developed to indicate the following relationships.

1. Training set: $TOOLSET0$ can't be used to deactivate the ORU. The $GDMS$ works well as a slave robot with the ORU.
   - Knowledge set: Dextrous manipulators can only use light toolsets.
2. Training set: The $JRMS$ can't be used near the $CARGO-BAY$. The APS can't be used with the ORU.

Knowledge set: None.

3. Training set: The $SPDM$ can't be used to attach the ORU. The $GDMS$ is particular good at attaching objects. The ORU can be reliably attached to $CARRIER-L$.

   Knowledge set: The ORU can't be attached to small carrier types. Grapplers can't be used to attach $CARRIER-M$ or $PALLET-L$. Pins can't be used to attach $CARRIER-L$ or $PALLET-M$.

4. Training set: The ATD can't move large carriers. The MT should be used to move large carriers.

   Knowledge set: Dextrous manipulators can't be used to move heavy carriers.

5. Training set: The $SPDM$ and $JRMS$ can't detach from $CARRIER-M$ or $CARRIER-L$. The FTS can't detach from $PALLET-M$ or $PALLET-L$.

   Knowledge set: None

6. Training set: Grapplers work well attaching the ORU to the MSC.

   Knowledge set: None

7. Training set: None

   Knowledge set: Transporters should not be used to stow the ORU. Only $PINS$, $GRAPPLER$, and $CLAMPS$ should be used to stow the ORU.

Since the training sets are limited, other symbol relationships may be developed besides the stated ones above. These "side-effects" will be demonstrated when prediction using these networks is presented.

The combination training set contains all the specific rules of training sets 1-7. Similarly, the combination knowledge set contains all the knowledge rules of knowledge sets 1-7. Therefore, there are 44 specific rules in the combination training set and 20 knowledge rules.

Training Set:

- *FTS DEACTUATE ORU NULL NULL TOOLSET1 → ORU IS-DEACTIVATED NULL POE: 0.92*
- *SPDM ACTUATE ORU NULL NULL TOOLSET2 → ORU IS-ACTIVATED NULL POE: 0.90*
- *FTS DEACTIVATE AWP NULL GDMS TOOLSET1 → AWP IS-DEACTIVATED NULL POE: 0.95*
- *FTS DEACTIVATE ORU NULL BLANK TOOLSET0 → ORU IS-DEACTIVATED NULL POE: 0.50*
- *JRMS OPERATE ORU NULL GDMS TOOLSET2 → ORU IS-ACTIVATED NULL POE: 0.92*

Knowledge Set:

- *< dex > NULL NULL NULL NULL TOOLSET3 → NULL IS-DEACTIVATED NULL*
- *< dex > NULL NULL NULL NULL TOOLSET4 → NULL IS-DEACTIVATED NULL*
- *< dex > NULL NULL NULL NULL TOOLSET5 → NULL IS-DEACTIVATED NULL*
- *< dex > NULL NULL NULL NULL TOOLSET6 → NULL IS-DEACTIVATED NULL*
- *< dex > NULL NULL NULL NULL TOOLSET3 → NULL IS-ACTIVATED NULL*
- *< dex > NULL NULL NULL NULL TOOLSET4 → NULL IS-ACTIVATED NULL*
- *< dex > NULL NULL NULL NULL TOOLSET5 → NULL IS-ACTIVATED NULL*
- *< dex > NULL NULL NULL NULL TOOLSET6 → NULL IS-ACTIVATED NULL*

Figure 6.6: Training set 1

Training Set:

- *JRMS DETACH ORU CARGO-BAY APS NULL →*
  ORU IS-DETACHED-FROM CARGO-BAY POE: 0.50
- *JRMS DETACH ORU MSC BLANK NULL →*
  ORU IS-DETACHED-FROM MSC POE: 0.95
- *GDMS DISCONNECT ORU TRUSS NULL PRYBAR →*
  ORU IS-DETACHED-FROM TRUSS POE: 0.75
- *MRMS DISCONNECT ORU TRUSS NULL SEPARATOR →*
  ORU IS-DETACHED-FROM TRUSS POE: 0.95

Knowledge Set: None

Figure 6.7: Training set 2

Training Set:

- *SPDM ATTACH ORU CARRIER-L FTS NULL →*
  ORU IS-ATTACHED-TO CARRIER-L POE: 0.40
- *GDMS ATTACH ORU CARRIER-L FTS NULL →*
  ORU IS-ATTACHED-TO CARRIER-L POE: 0.97
- *SPDM ATTACH PINS-H CARRIER-L FTS NULL →*
  PINS-H IS-ATTACHED-TO CARRIER-L POE: 0.84
- *SPDM POSITION ORU CARGO-BAY NULL NULL →*
  ORU IS-AT CARGO-BAY POE: 0.82
- *MRMS CONNECT ORU PALLET-L NULL PINS-H →*
  ORU IS-ATTACHED-TO PALLET-L POE: 0.97
- *FTS FASTEN ORU CARRIER-L BLANK GRAPPLER-M →*
  ORU IS-ATTACHED-TO CARRIER-L POE: 0.85
- *GDMS CONNECT PALLET-M CARRIER-M NULL PINS-M →*
  PALLET-M IS-ATTACHED-TO CARRIER-M POE: 0.95
- *FTS CONNECT PALLET-M CARRIER-M NULL PINS-H →*
  PALLET-M IS-ATTACHED-TO CARRIER-M POE: 0.97

Knowledge Set:

- *NULL NULL ORU CARRIER-S NULL NULL →*
  NULL IS-ATTACHED-TO NULL
- *NULL NULL ORU PALLET-S NULL NULL →*
  NULL IS-ATTACHED-TO NULL
- *NULL NULL NULL PALLET-L NULL < grappler > →*
  NULL IS-ATTACHED-TO NULL
- *NULL NULL NULL PALLET-M NULL < pins > →*
  NULL IS-ATTACHED-TO NULL
- *NULL NULL NULL CARRIER-M NULL < grappler > →*
  NULL IS-ATTACHED-TO NULL
- *NULL NULL NULL CARRIER-L NULL < pins > →*
  NULL IS-ATTACHED-TO NULL

Figure 6.8: Training set 3

C-#3

Training Set:

- *JRMS PLACE CARRIER-S MSC BLANK NULL →*
  CARRIER-S IS-AT MSC POE: 0.95
- *FTS PLACE PALLET-S TRUSS BLANK NULL →*
  PALLET-S IS-AT TRUSS POE: 0.97
- *FTS PLACE TOOLSET0 MSC BLANK NULL →*
  TOOLSET0 IS-AT MSC POE: 0.95
- *SPDM PLACE GLUEGUN TRUSS BLANK NULL →*
  GLUEGUN IS-AT TRUSS POE: 0.97
- *ATD PLACE PALLET-L CARGO-BAY FTS NULL →*
  PALLET-L IS-AT CARGO-BAY POE: 0.43
- *ATD PLACE PALLET-S TRUSS FTS NULL →*
  PALLET-S IS-AT TRUSS POE: 0.87
- *ATD POSITION CARRIER-M TRUSS NULL NULL →*
  CARRIER-M IS-AT TRUSS POE: 0.89
- *ATD POSITION CARRIER-L TRUSS NULL NULL →*
  CARRIER-L IS-AT TRUSS POE: 0.40
- *APS POSITION PALLET-M CARGO-BAY NULL NULL →*
  PALLET-M IS-AT CARGO-BAY POE: 0.92
- *MT TRANSPORT CARRIER-L TRUSS NULL NULL→*
  CARRIER-L IS-AT TRUSS POE: 0.95
- *MT TRANSPORT PALLET-L CARGO-BAY NULL NULL →*
  PALLET-L IS-AT CARGO-BAY POE: 0.92

Knowledge Set:

- *< dex > NULL PALLET-L NULL NULL NULL →*
  NULL IS-AT NULL
- *< dex > NULL PALLET-M NULL NULL NULL →*
  NULL IS-AT NULL
- *< dex > NULL CARRIER-L NULL NULL NULL →*
  NULL IS-AT NULL
- *< dex > NULL CARRIER-M NULL NULL NULL →*
  NULL IS-AT NULL

Figure 6.9: Training set 4

Training Set:

- *SPDM DETACH ORU CARRIER-M BLANK NULL →*
  ORU IS-DETACHED-FROM CARRIER-M POE: 0.45
- *SPDM DETACH ORU CARRIER-L BLANK NULL →*
  ORU IS-DETACHED-FROM CARRIER-L POE: 0.50
- *SPDM DETACH GRAPPLER-M PALLET-L BLANK NULL →*
  GRAPPLER-M IS-DETACHED-FROM PALLET-L POE: 0.95
- *JRMS DISCONNECT ORU CARRIER-M NULL PRYBAR →*
  ORU IS-DETACHED-FROM CARRIER-M POE: 0.55
- *JRMS DISCONNECT ORU CARRIER-L NULL PRYBAR →*
  ORU IS-DETACHED-FROM CARRIER-L POE: 0.52
- *JRMS DISCONNECT GRAPPLER-L PALLET-M NULL PRYBAR →*
  GRAPPLER-L IS-DETACHED-FROM PALLET-M POE: 0.98
- *FTS DISCONNECT ORU PALLET-L NULL SEPARATOR →*
  ORU IS-DETACHED-FROM PALLET-L POE: 0.43
- *FTS DISCONNECT ORU PALLET-M NULL SEPARATOR →*
  ORU IS-DETACHED-FROM PALLET-M POE: 0.52
- *FTS DISCONNECT ORU CARRIER-L NULL SEPARATOR →*
  ORU IS-DETACHED-FROM CARRIER-L POE: 0.95
- *GDMS DISCONNECT PALLET-M CARRIER-L NULL PRYBAR →*
  PALLET-M IS-DETACHED-FROM CARRIER-L POE: 0.87
- *MRMS DETACH PINS-L PALLET-M BLANK NULL →*
  PINS-L IS-DETACHED-FROM PALLET-M POE: 0.95
- *GDMS DETACH PALLET-S CARRIER-M APS NULL →*
  PALLET-S IS-DETACHED-FROM CARRIER-M POE: 0.90
- *MRMS DISCONNECT PINS-H PALLET-M NULL SEPARATOR →*
  PINS-H IS-DETACHED-FROM PALLET-M POE: 0.92

Knowledge Set: None

Figure 6.10: Training set 5

Training Set:

- *FTS ATTACH ORU MSC BLANK NULL →*
  ORU IS-ATTACHED-TO MSC POE: 0.40
- *FTS CONNECT ORU MSC NULL GRAPPLER-M →*
  ORU IS-ATTACHED-TO MSC POE: 0.95
- *SPDM FASTEN ORU MSC FTS GRAPPLER-H →*
  ORU IS-ATTACHED-TO MSC POE: 0.95

Knowledge Set: None

Figure 6.11: Training set 6

Training Set: None
Knowledge Set:

- *< trans > NULL ORU NULL NULL NULL →*
  NULL IS-STOWED-AT NULL
- *NULL NULL ORU NULL NULL < fix − other > →*
  NULL IS-STOWED-AT NULL

Figure 6.12: Training set 7

6.4.3 Training results

This section presents the results of training the ARM network on each of the training sets, and the result of training the network on the combined training sets.

The following training constants are used in the case study:

- $0 \leq w_{ij} \leq 1$, the bounds on the weights.
- $\epsilon = 0.05$, the gradient step size for training sets 1 - 4 and 6. The gradient step size for training sets 5 and the combination set is $\epsilon = 0.025$.
- $Differror = 0.25$, the error band for rule selection.
- $\Theta_1 = 10^{-9}$, the minimum gradient before stopping.
- $\Theta_2 = 5 \times 10^{-3}$, the desired accuracy of each rule in the training set.
- $\Theta_5 = 1 \times 10^{-4}$, the minimum weight before pruning is allowed.
- $\alpha_1 = 1 \times 10^{-4}$, the first order forcing function constant.
- $\alpha_2 = 4 \times 10^{-3}$, the second order forcing function constant.

Since the number of rules in training set 5 and the combination set is large, a smaller step size has to be used during gradient descent.

Table 6.1 presents the results of training. As evident from the data in this table, each set was trained very accurately on the network. In fact, the maximum error of any of the rules in any of the training sets was less than 0.01 of the acceptable error. Training set 3 generated 10 second order nodes, of which 6 were removed through the pruning process. The combination training set generated 20 second order nodes of which 14 were pruned. It is also evident from the data that larger training sets required more gradient iterations, which is expected.

The results of the training show that it is quite difficult to generate higher order nodes. Since the number of first order connections available to a specific rule

Table 6.1: Results of case study training sets

Training Set	*Iterations*	*Max* $(G_\gamma)^2$	$\sum_\gamma (G_\gamma)^2$
1	184	0.9×10^{-9}	1.6×10^{-9}
2	90	0.2×10^{-9}	0.5×10^{-9}
3	404	0.8×10^{-9}	1.7×10^{-9}
4	357	1.1×10^{-9}	2.8×10^{-9}
5	1385	0.4×10^{-9}	1.4×10^{-9}
6	88	0.0×10^{-9}	0.1×10^{-9}
7	-	-	-
Comb.	2398	1.9×10^{-9}	3.6×10^{-9}

is large, a specific rule in the training set can often place the blame for a low POE value on one of its connections that is not used by other rules in the training set. This inhibits the development of a higher order relationship. Using a larger training set with more overlapping sets of symbols would reduce the number of rules that have first order connections that can be falsely blamed for low POE values. This would force the training procedure to generate more higher order nodes. It is also possible to generate more higher order nodes if the maximum weight allowed on a connection is decreased by reducing the allowable blame for first order connections.

Another method for developing more higher order relationships is to increase the value of α_1, in the first order forcing function (3.23), past its maximum allowable value (3.35). This would reduce the chance that the desired accuracy of a specific rule could be achieved in first order connections, and would lead to the development of higher order nodes.

6.4.4 Examples of Prediction

After each training, the ARM was evaluated for its prediction capabilities on the given training set. To do this, a set of specific rules was constructed that tested the desired relationships of the training set (as described in section 6.4.2). Each rule was asserted on the network. The POE value of each asserted rule was then

examined in terms of the desired relationships. The base probability value for an untested specific rule was set at 0.80.

Figures 6.13 and 6.14 show sample prediction rules for training sets 1 and 3, respectively. Although each network was tested with a larger set of rules, these figures provide a good demonstration of prediction with the ARM.

Examining figure 6.13, we see that test rules 1 - 3 are members of the training set, and were accurately represented when asserted on the network. Further, the cf is 1.0, denoting training set rules. Rule 4 in this figure demonstrates that the network was able to produce a low POE value with high confidence for a rule that dictates the deactivation of the ORU with $TOOLSET0$. Rules 5, 6 and 7 also show inhibition between these symbols, but demonstrate another relationship developed through training. The training set has allowed inhibition to be created between the nodes $DEACTUATE$ in the robotic action and ORU in the effect. Although this may be true in the real world, this relationship is something that we (the user) did not foresee, but was evidently present in training set 1. The cf values for these rules are somewhat low, denoting both untested connections and reduced overlap with rules in the training set. Rule 8 demonstrates that a low POE value is assigned to a rule that is provided by knowledge set 1, which contains the list of symbol combinations that should be avoided. The final rule presents the prediction of an untested rule with a high POE value and a high cf.

Figure 6.14 also demonstrates the storage and prediction capabilities of the ARM. Rules 1 - 3 are members of training set 3 and were accurately stored. Rules 4, 5 and 6 demonstrate that the $SPDM$ can move the ORU, and that the $SPDM$ and can attach $PALLET - M$ fairly well; however, the $SPDM$ cannot attach the ORU. This is a higher order relationship and requires second order nodes. Rule 7 shows that the $GDMS$ performs well at achieving an attachment. Rules 8 and 9 contain symbol combinations that are disallowed by the knowledge set and, thereby,

1. *FTS DEACTUATE ORU NULL NULL TOOLSET1 →*
   ORU IS-DEACTIVATED NULL
   E = 0.083394; POE = 0.919989; cf = 1.000000

2. *FTS DEACTIVATE ORU NULL BLANK TOOLSET0 →*
   ORU IS-DEACTIVATED NULL
   E = 0.693139; POE = 0.500004; cf = 1.000000

3. *JRMS OPERATE ORU NULL GDMS TOOLSET2 →*
   ORU IS-ACTIVATED NULL
   E = 0.083382; POE = 0.920000; cf = 1.000000

4. *JRMS DEACTIVATE ORU NULL BLANK TOOLSET0 →*
   ORU IS-DEACTIVATED NULL
   E = 0.672337; POE = 0.510514; cf = 0.848528

5. *JRMS DEACTIVATE ORU NULL GDMS TOOLSET0 →*
   ORU IS-DEACTIVATED NULL
   E = 0.453924; POE = 0.635131; cf = 0.734847

6. *SPDM DEACTUATE ORU NULL NULL TOOLSET0 →*
   ORU IS-DEACTIVATED NULL
   E = 0.293932; POE = 0.745327; cf = 0.661438

7. *FTS DEACTIVATE ORU NULL GDMS TOOLSET1 →*
   ORU IS-DEACTIVATED NULL
   E = 0.251358; POE = 0.777744; cf = 0.774597

8. *FTS DEACTUATE ORU NULL NULL TOOLSET3 →*
   ORU IS-DEACTIVATED NULL
   E = 1.115272; POE = 0.327826; cf = 0.750000

9. *JRMS DEACTUATE ORU NULL NULL TOOLSET1 →*
   ORU IS-DEACTIVATED NULL
   E = 0.062591; POE = 0.939327; cf = 0.810093

Figure 6.13: Prediction using training set 1

1. *SPDM ATTACH ORU CARRIER-L FTS NULL →*
   ORU IS-ATTACHED-TO CARRIER-L
   E = 0.916282; POE = 0.400004; cf = 1.000000

2. *MRMS CONNECT ORU PALLET-L NULL PINS-H →*
   ORU IS-ATTACHED-TO PALLET-L
   E = 0.030459; POE = 0.970000; cf = 1.000000

3. *FTS CONNECT PALLET-M CARRIER-M NULL PINS-H →*
   PALLET-M IS-ATTACHED-TO CARRIER-M
   E = 0.030459; POE = 0.970000; cf = 1.000000

4. *SPDM CONNECT ORU CARRIER-L NULL GRAPPLER-M →*
   ORU IS-ATTACHED-TO CARRIER-L
   E = 0.743413; POE = 0.475488; cf = 0.748331

5. *SPDM POSITION ORU PALLET-L NULL NULL →*
   ORU IS-AT PALLET-L
   E = 0.246296; POE = 0.781691; cf = 0.612372

6. *SPDM ATTACH PALLET-M CARRIER-M FTS NULL →*
   PALLET-M IS-ATTACHED-TO CARRIER-M
   E = 0.190737; POE = 0.826350; cf = 0.489898

7. *GDMS FASTEN ORU CARRIER-L FTS GRAPPLER-M →*
   ORU IS-ATTACHED-TO CARRIER-L
   E = 0.097969; POE = 0.906677; cf = 0.816497

8. *GDMS FASTEN ORU CARRIER-M FTS GRAPPLER-M →*
   ORU IS-ATTACHED-TO CARRIER-M
   E = 1.128829; POE = 0.323412; cf = 0.490653

9. *GDMS FASTEN ORU CARRIER-S FTS GRAPPLER-M →*
   ORU IS-ATTACHED-TO CARRIER-S
   E = 1.150086; POE = 0.316610; cf = 0.430331

Figure 6.14: Prediction using training set 3

have low POE values.

Overall, the predictive POE values provided by the ARM for each of the training sets showed that the ARM successfully represented the desired relationships. Occasionally, the ARM developed unexpected symbolic relationships as well. These additional relationships are present in the training set, however, and demonstrate that the ARM can point out subtle relationships in a training set that may have gone previously unnoticed by a user.

Of course, the user can explicitly discover which symbolic relationships were formed during training by examining the connection weights of the ARM network, since there is a one-to-one mapping between symbols and nodes. If a relationship exists in the connections that the user did not expect, he can specifically create new test situations in the robotic environment to examine these relationships. After testing, the training set can be updated with the results of the new experiments, and the ARM network can be accurately retrained with the modified data. Therefore, the ARM also functions as a feedback mechanism that the user can exploit to determine and test relationships that may not have been previously realized.

Figure 6.15 shows each of untested specific rules in Figures 6.13 and 6.14 asserted on the ARM network trained with training sets 1 - 7 combined. Notice that the probability values have changed for many of the rules. This reflects that the increased training has affected the weighted connections to make them more representative of the combined training set.

This figure also point out a flaw in the cf measure we have chosen. It would be preferable if the cf of each of the specific rules increased to indicate that the ARM is "more confident" of the POE value with more training. This is not one of the properties of our current cf measure, as indicated by cf values that are nearly the same as those in the previous figures. If desired, a new cf measure can easily be created by the user that also takes into account the number of times a connection

is used by different rules. The more times each connection of an untested rule is used by specific rules in the training set, the higher the cf value would be for the untested rule.

1. *JRMS DEACTIVATE ORU NULL BLANK TOOLSET0* →
   ORU IS-DEACTIVATED NULL
   E = 0.692621 POE = 0.500263 cf = 0.848528

2. *JRMS DEACTIVATE ORU NULL GDMS TOOLSET0* →
   ORU IS-DEACTIVATED NULL
   E = 0.562452 POE = 0.569810 cf = 0.734847

3. *SPDM DEACTUATE ORU NULL NULL TOOLSET0* →
   ORU IS-DEACTIVATED NULL
   E = 0.402713 POE = 0.668504 cf = 0.661438

4. *JRMS DEACTUATE ORU NULL NULL TOOLSET1* →
   ORU IS-DEACTIVATED NULL
   E = 0.082856 POE = 0.920484 cf = 0.810093

5. *SPDM CONNECT ORU CARRIER-L NULL GRAPPLER-M* →
   ORU IS-ATTACHED-TO CARRIER-L
   E = 0.642305 POE = 0.526078 cf = 0.748331

6. *SPDM POSITION ORU PALLET-L NULL NULL* →
   ORU IS-AT PALLET-L
   E = 0.163904 POE = 0.848823 cf = 0.612372

7. *SPDM ATTACH PALLET-M CARRIER-M FTS NULL* →
   PALLET-M IS-ATTACHED-TO CARRIER-M
   E = 0.402055 POE = 0.668944 cf = 0.516398

8. *GDMS FASTEN ORU CARRIER-L FTS GRAPPLER-M* →
   ORU IS-ATTACHED-TO CARRIER-L
   E = 0.000000 POE = 1.000000 cf = 0.816497

Figure 6.15: Prediction using the combined training set

6.5 Associative Recall of Robotic Actions

This section describes the associative recall experiments performed on the ARM trained with training sets 1 - 7 combined. The purpose of the experimentation is to find the optimal robotic action, or set of near-optimal robotic actions, that have high POE values for effects stated in the goal plan presented in section 6.2. The high POE robotic actions form the planning steps that must be carried out in the world to achieve the stated goal.

6.5.1 Representation of Nodes

As mentioned in Chapter 4, the GA representation chosen for ARM nodes can affect the efficiency of the GA search. Under the Principle of Meaningful Building Blocks, it is known that short-order schemata should possess meaningful, semantic information about the problem domain. Using this as a guideline, a binary representation is developed from the characteristics of the agents in the world. Before presenting the node representation, however, several points must be made.

First, it must be stated that the representation chosen is not "optimal" by some criteria. An optimal representation would only provide best-case performance of the GA for the given problem. Instead, a somewhat inefficient, heuristic representation is chosen. If the GA performs well with an inefficient representation, the results are much more impressive, since it demonstrates that the GA is a robust technique for associative recall across various degrees of representational efficiency.

Second, for a given desired effect, there are a known set of general rules that are capable of generating the effect. The general rules denote the set of allowable agents and actions that can be used to perform the robotic action. This provides the user with a choice. The user can allow either the representation to encompass all possible actions and agents, or the user can restrict the representation to include only actions and agents that can possibly achieve the effect. Either method will

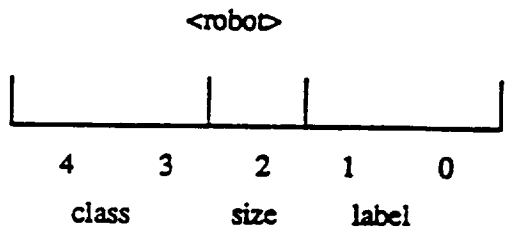
allow the GA to perform correct associative recall, but the second method reduces the size of the search space, leading to a more efficient recall process. Therefore, the second method is chosen for representation in this case study.

By using the general rules as a "prefilter" for allowable agents and actions, the second method somewhat reduces the need for general rules as inhibitory links in the ARM network. From the structure of the general rules, however, we do see that different actions that achieve the same effect often require different classes of agent symbols as part of the robotic action. Therefore, the general rules are still required by the ARM to restrict the agent symbols that can be used with each of the allowable actions.

Given the above discussion, it is possible to present the representation of the set of nodes that form a robotic action in the ARM network. The representation of each node comes directly from the agent class hierarchy presented in Appendix B with some semantic information about the agents information provided in Appendix A.

Each node encodes information about its class as well as its features in binary form, as presented in Figures 6.16 and 6.17. The first of these examples displays the representation of agents of the class $< robot >$. Five bits are used for the representation of 15 agents, which is inefficient, but allows meaningful building blocks to occur. The second of these figures displays a more efficient representation for objects of the class $< fix >$. In this figure, four bits are used to encode 12 objects. Using a similar strategy, each class of symbols in the TAM is encoded.

In addition to representation of the symbols of each class, the GA must also encode the symbols $NULL$, and $BLANK$. The second figure demonstrates the encoding of a $BLANK$ node. Binary values that are not assigned to TAM symbols, or the $BLANK$ or $NULL$ symbols are assigned to the symbol XXX which denotes an invalid representation.



BITS:

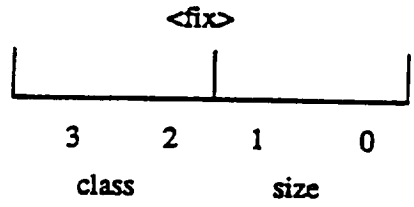
4-3: 00 = <dex>, 01 = <pos>, 10 = <trans>

2: 0 = light, 1 = heavy

ENCODING:

00000: EVA	00001: JRMS	00010: FTS	00011: SPDM
00100: GDMS	00101: MRMS		
01000: SRMS	01001: SSRMS		
01100: ATD	01101: APS		
10000: MMU	10001: CETA	10010: MT	
10100: OMV	10101: ORBITER		

Figure 6.16: Representation example for agents in < *robot* > class



BITS:

3-2: 00 = <pins>, 01 = <clamps>, 10 = <grappler>,
11 = <fix-other>
1-0: 00 = heavy, 01 = medium, 10 = light

ENCODING:

0000: PINS-H	0001: PINS-M	0010: PINS-L
0100: CLAMP-H	0101: CLAMP-M	0110: CLAMP-L
1000: GRAPPLER-H	1001: GRAPPLER-M	1010: GRAPPLER-L
1100: WELDER	1101: BOLTER	1110: GLUEGUN

1111: BLANK

Figure 6.17: Representation example for agents in $< fix >$ class

For a given desired effect, a set of action symbols are allowed. These actions symbols are assigned binary values between 0 and the number of allowed actions.

Since we know that the direct and indirect objects of the robotic action must be the same as the objects in the desired effect, they need not be explicitly represented in the search string. Each GA member, therefore, consists of a concatenation of binary strings for nodes on the *ACTOR, ACTION, SLAVE,* and *TOOL* levels. For example, to represent the robotic action of the specific rule

$$MT\ STOW\ ORU\ MSC\ FTS\ GRAPPLER-M\ \rightarrow$$

$$ORU\ IS-STOWED-AT\ MSC$$

the GA member would be:

1001000101001

where the first 5 bits 10010 encode MT, the next bit, 0, encodes $STOW$, the next three bits 010 encode FTS, and the last 4 bits encode $GRAPPLER-M$. ORU and MSC are not encoded because they are implicitly stated by the desired effect.

The GA can produce binary strings that do not map to TAM symbols, through its selection and recombination procedure. These binary representations map to the XXX symbol described above. To inhibit the GA from developing invalid representations, any binary string that maps to an XXX node is assigned a very low POE value. This is called the "Penalty Method" for avoiding bad regions in the domain of the GA. Over repeated iterations, the GA should be able to avoid invalid representations by not promoting the building blocks that map to XXX nodes.

6.5.2 The GA search process

Given the representation of nodes, the GA search can be summarized as follows.

Given a Desired Effect,

1. The set of allowable nodes for each level is determined.

2. For each level, the smallest class that subsumes all the allowable node symbols is found.

3. The length of a GA member is determined by summing the lengths of the required binary representations of the class at each level. Let us call this length l.

4. A random population of binary strings of length l is created. Based on the results of section 4.3, we choose a population of 60 members with an immigration rate of 2 members/iteration. We use a steady state GA.

5. The cost of a member is the Energy value of the member when mapped to TAM symbols and asserted on the network. The fitness of each member is found using equation (4.6).

6. The GA search process is run. The user may include the speciation algorithm in the GA process, if desired. The process concludes when either of the following occurs.

   (a) The minimum cost member is found. This is the optimal robotic action, and has the largest POE for the given desired effect.

   (b) The best member in the population has a cost below a user-provided threshold. This corresponds to a near optimal robotic action.

7. Depending on the inclusion of speciation, the GA responds with either

(a) the best robotic action found and the corresponding POE and cf values

or

(b) a set of near-optimal robotic actions with corresponding POE value and cf values.

6.5.3 Embodying planning constraints into the recall process

It would be helpful if some features of the planning process could be incorporated into the associative recall process. This would allow the planning procedure influence the search, and select or avoid certain robotic actions. Examples of helpful features are:

1. The ability to inhibit the inclusion of particular agents in the robotic action. This is useful when particular resources are not currently available to achieve the desired effect, or the resources are otherwise constrained.

2. The ability to force the inclusion of particular agents in the robotic action. This allows the planner to force a particular resource to be used.

3. The ability to leave part of the robotic action unspecified. This allows the planner to leave rule variables uninstantiated.

4. The ability to leave part of the desired effect unspecified, and have it specified by the recall process. This is helpful if the planner wants to find the best direct or indirect object to be used to accomplish the desired effect.

One method for inhibiting the inclusion of a particular agent is to remove it from the class of allowable nodes for that level. In effect, this maps the binary string representing the agent to the XXX symbol, and yields a low POE value for any robotic action containing that agent. Other methods can also be used.

On the other hand, forcing the inclusion of a particular agent can be achieved by removing the portion of the binary string representing the agent's level from the

GA representation of the robotic action. The agent is is explicitly asserted on the network with each robotic action. This similar to the assertion of direct or indirect objects on the ARM network.

The planner may wish to leave part of the robotic action unspecified because it does not want to instantiate a particular variable. This can be achieved by removing the binary string representing the class of symbols for the variable's level from the GA representation of the robotic action. By not asserting any node on an agent level of the network, the POE value for each robotic action is now an upper bound of possible POE values.

The planner may want the recall process to dictate the direct object or indirect object to use. For example, the planner may provide the ARM with the desired effect:

$$ORU\ \ IS-ATTACHED-TO\ \ <carrier>$$

This would force the recall process to find the carrier, and corresponding robotic action, which produces the optimal POE value. This is particularly useful in an environment with many similar resources available. The planner wishes to select the resource that has the highest probability of leading to a successful plan.

This can be accomplished by finding the substring in the robotic action that corresponds to the given variable, determining its symbolic mapping, and asserting it on the input nodes. This still constrains the direct and indirect objects to be the same in the robotic action and effect portions of the network.

6.5.4 Experimental procedure

Associative recall experiments were performed on the ARM network trained with training sets 1 - 7 combined. The following desired effects were used for each experiment, corresponding to the seven planning steps:

1. *ORU IS-DEPLOYED NULL*
2. *ORU IS-DETACHED-FROM TRUSS*
3. *ORU IS-ATTACHED-TO < carrier >*
4. *< carrier > IS-AT MSC*
5. *ORU IS-DETACHED-FROM < carrier >*
6. *ORU IS-ATTACHED-TO MSC*
7. *ORU IS-STOWED-AT MSC*

We can see that desired effects 3, 4 and 5 require the associative recall technique to specify a carrier. The other desired effects use the standard recall procedure.

As demonstrated by the seven steps above, the ARM is used in this case study to provide robotic actions for an ordered list of effects that have already been provided by an external planning system. It is important to remember that the ARM can also be used to limit the search space when developing these planning steps, if combined with a search strategy such as Means-End Analysis.

6.5.5 Experimental results: Efficiency of the GA

The robotic action with the highest POE value (optimal robotic action) is first determined for each desired effect through experimentation with the ARM network. Each effect is then asserted on the network, and 500 experiments are run to determine the average number of robotic actions that had to be evaluated to find

Table 6.2: Results of case study associative recall

Recall exp.	*Avg*	*Pos*	*Siz*	*Spe*
1	340	1584	4096	-
2	364	792	2048	-
3	2198	20592	65536	2643
4	326	5940	32768	857
5	1148	4752	16384	1445
6	902	3432	8192	845
7	322	1170	4096	-

the optimal robotic action for the given desired effect. This is the same measure used in Chapter 4, the number of function evaluations. A subset of the desired effects are then chosen to test the speciation procedure

Table 6.2 presents the results of the experiments on the ARM network. This table shows the average number of function evaluations that were required to find the optimal robotic action for each desired effect (Avg), the total number of possible robotic actions for the given effect (Pos), the size of the binary search space (Siz), and the average number of function evaluations required to find the optimal robotic action when using speciation (Spe). It is important to realize that the binary search space is larger than the number of possible robotic actions due to representational inefficiency in the encoding of robotic actions.

It is also important to note that each time a robotic action was evaluated, it was counted, even if the same robotic action had been evaluated earlier in the GA. For each GA run, therefore, some points were evaluated many times, with each evaluation adding to the number of function evaluations for that GA run.

As demonstrated by experiments 1, 5, 6 and 7, the GA was able to find the optimal robotic action by evaluating about $\frac{1}{4}$ of the possible number of robotic actions. Experiments 3 and 4, which had the largest search spaces, show a significant reduction in the search time, each evaluating less than $\frac{1}{10}$ of the number of possible

search points. Experiment 2 shows the worst performance, requiring the evaluation of about $\frac{1}{2}$ of the possible number of robotic actions. Based on this information, the GA seems to perform efficiently on all the optimization tasks, except those which possess a small domain.

When compared to the total size of the search space, the GA operated very efficiently, demonstrating that the Penalty Method was effective for avoiding invalid robotic actions.

From this data, it seems that the chosen GA requires a minimum of about 300 evaluations to find the optimum value, even for the smaller search spaces. This would indicate that for smaller search spaces:

- a smaller, more exploitive GA should be used, or
- simple enumeration of all robotic actions may be quite effective.

It is interesting to note that in three of the four experiments that included speciation, the average number of function evaluations increased less than 25 percent. In experiment 4, the number of evaluations more than doubled, but was still small compared to the possible number of robotic actions. This indicates that the speciation technique we chose leads to a less efficient GA, though one whose efficiency is still acceptable for finding optimal robotic actions.

6.5.6 Experimental results: Optimal robotic actions

The GA found the following optimal robotic actions for the seven planning steps of the goal plan using the ARM network trained on training sets 1 - 7 combined.

1. Desired effect: *ORU IS-DEACTIVATED NULL*
   Robotic Action: *GDMS DEACTUATE ORU NULL NULL TOOLSET1*
   E = 0.0828; POE = 0.92; cf = 0.81

2. Desired effect: *ORU IS-DETACHED-FROM TRUSS*

   Robotic Action: *MRMS DETACH ORU TRUSS BLANK NULL*

   E = 0.0482; POE = 0.95; cf = 0.75

3. Desired effect: *ORU IS-ATTACHED-TO < carrier >*

   Robotic Action: *GDMS FASTEN ORU CARRIER-L FTS GRAPPLER-M*

   E = 0.000; POE = 0.99; cf = 0.

4. Desired effect: *< carrier > IS-AT MSC*

   Robotic Action: *JRMS PLACE CARRIER-S MSC BLANK NULL*

   E = 0.051; POE = 0.95; cf = 1.0

5. Desired effect: *ORU IS-DETACHED-FROM < carrier >*

   Robotic Action: *GDMS DISCONNECT ORU CARRIER-L NULL SEPARATOR*

   E = 0.0389; POE = 0.96; cf = 0.89

6. Desired effect: *ORU IS-ATTACHED-TO MSC*

   Robotic Action: *GDMS FASTEN ORU MSC FTS GRAPPLER-H*

   E = 0.0157; POE = 0.98; cf = 0.89

7. Desired effect: *ORU IS-STOWED-AT MSC*

   Robotic Action: *GDMS STOW ORU MSC FTS PINS-H*

   E = 0.1610; POE = 0.85; cf = 0.38

Many of the planning steps use the same actors, which allows for a conservation of resources. This set of robotic actions will not achieve the goal state, however, due to steps 3, 4 and 5. These robotic actions do not use the the same carrier, so the *ORU* can't be moved to the *MSC* successfully using these steps. To find a common carrier that can be used in all three steps, a GA with speciation is applied.

The results of a sample speciation run yield the following robotic action alternatives for steps 3, 4 and 5:

Desired Effect: *ORU IS-ATTACHED-TO < carrier >*

- *GDMS FASTEN ORU CARRIER-L FTS GRAPPLER-M*
  E = 0.0000; POE = 0.99; cf = 0.81
- *GDMS FASTEN ORU CARRIER-L FTS GRAPPLER-H*
  E = 0.0160; POE = 0.98; cf = 0.79
- *GDMS FASTEN ORU CARRIER-L FTS GLUEGUN*
  E = 0.0478; POE = 0.95; cf = 0.75
- *GDMS FASTEN ORU CARRIER-L FTS GRAPPLER-L*
  E = 0.0478; POE = 0.95; cf = 0.75
- *MRMS FASTEN ORU CARRIER-L FTS GLUEGUN*
  E = 0.0669; POE = 0.94; cf = 0.62
- *MRMS FASTEN ORU PALLET-L FTS GLUEGUN*
  E = 0.0891; POE = 0.91; cf = 0.60

Desired Effect: *< carrier > IS-AT MSC*

- *JRMS PLACE CARRIER-S MSC BLANK NULL*
  E = 0.0512; POE = 0.95; cf = 1.0
- *JRMS POSITION CARRIER-S MSC NULL NULL*
  E = 0.0527; POE = 0.95; cf = 0.79
- *MT POSITION CARRIER-M MSC NULL NULL*
  E = 0.0602; POE = 0.94; cf = 0.44

- *MT TRANSPORT CARRIER-M MSC NULL NULL*

  E = 0.0637; POE = 0.94; cf = 0.28

- *APS POSITION CARRIER-M MSC NULL NULL*

  E = 0.0641; POE = 0.94; cf = 0.44

- *MT TRANSPORT CARRIER-L MSC NULL NULL*

  E = 0.0831; POE = 0.92; cf = 0.58

Desired Effect: *ORU IS-DETACHED-FROM < carrier >*

- *GDMS DISCONNECT ORU CARRIER-L NULL SEPARATOR*

  E = 0.0390; POE = 0.96; cf = 0.89

- *MRMS DETACH ORU PALLET-M SSRMS NULL*

  E = 0.0604; POE = 0.94; cf = 0.57

- *MRMS DETACH ORU PALLET-M BLANK NULL*

  E = 0.0605; POE = 0.94; cf = 0.73

- *JRMS DETACH ORU PALLET-M SSRMS NULL*

  E = 0.0606; POE = 0.94; cf = 0.57

- *JRMS DETACH ORU PALLET-M SRMS NULL*

  E = 0.0606; POE = 0.94; cf = 0.57

- *MRMS DETACH ORU PALLET-L ATD NULL*

  E = 0.0692; POE = 0.93; cf = 0.56

The speciation process did an excellent job of developing sets of robotic action alternatives. Using these alternatives, the planner can select common carrier types and can form the following plan.

1. Desired effect: *ORU IS-DEACTIVATED NULL*

   Robotic Action: *GDMS DEACTUATE ORU NULL NULL TOOLSET1*

   E = 0.0828; POE = 0.92; cf = 0.81

2. Desired effect: *ORU IS-DETACHED-FROM TRUSS*

   Robotic Action: *MRMS DETACH ORU TRUSS BLANK NULL*

   E = 0.0482; POE = 0.95; cf = 0.75

3. Desired effect: *ORU IS-ATTACHED-TO < carrier >*

   Robotic Action: *GDMS FASTEN ORU CARRIER-L FTS GRAPPLER-M*

   E = 0.000; POE = 0.99; cf = 0.

4. Desired effect: *< carrier > IS-AT MSC*

   Robotic Action: *MT TRANSPORT CARRIER-L MSC NULL NULL*

   E = 0.0831; POE = 0.92; cf = 0.58

5. Desired effect: *ORU IS-DETACHED-FROM < carrier >*

   Robotic Action: *GDMS DISCONNECT ORU CARRIER-L NULL SEPARATOR*

   E = 0.0389; POE = 0.96; cf = 0.89

6. Desired effect: *ORU IS-ATTACHED-TO MSC*

   Robotic Action: *GDMS FASTEN ORU MSC FTS GRAPPLER-H*

   E = 0.0157; POE = 0.98; cf = 0.89

7. Desired effect: *ORU IS-STOWED-AT MSC*

   Robotic Action: *GDMS STOW ORU MSC FTS PINS-H*

   E = 0.1610: POE = 0.85; cf = 0.38

If desired, the planner may wish to reduce the number of agents used in the plan, by replacing the *MRMS* in step 2 with the *GDMS*. The planner can query

the network with the specific rule

$$GDMS\ DETACH\ ORU\ TRUSS\ BLANK\ NULL \rightarrow$$

$$ORU\ IS-DETACHED-FROM\ TRUSS$$

and determine that E = 0.0640, POE = 0.94, and cf = 0.75. The high POE and cf values determine that this robotic action will lead to a successful planning step, so the substitution can occur.

This example, therefore, demonstrates the ability of the ARM to develop successful plans given a set of desired effects.

6.6 Conclusions

This chapter presents a case study based on the NASA Flight Telerobotic Servicer Task Analysis Methodology. The results presented in this chapter provide the following evaluation of the ARM model.

1. The ability to represent general rules. The ARM is able to represent a set general rules that contains groups of rules that lead to the same effect, and rules that lead to multiple effects.

2. The ability to accurately store specific rules and corresponding POE values from the training set. The training algorithm produces a very efficient representation of each rule in the training set. Higher order nodes are created, and pruned when necessary. Difficulty in developing higher order nodes can occur if the rules in the training set do not overlap greatly, in which case most of the for low POE values is placed on first order connections.

3. The ability to predict POE values for untested specific rules. The ARM is able to develop the representations given in the training set. Subtle representations that the user may not see are also developed and may help the user to

create new robotic tests. The development of symbolic relationships leads to reasonable predicted POE values for untested specific rules.

4. The speed of associative recall of the optimal robotic action for each desired effect of the plan. The GA is shown to be efficient as a recall procedure for medium and large domain sizes. For smaller domains, the chosen GA is not as efficient.

5. The speed of associative recall of a set of near-optimal robotic actions for each desired effect of the plan. Speciation results demonstrate some drop in efficiency in the GA algorithm, but still provide acceptable results. Speciation does an excellent job producing sets of high POE robotic actions.

CHAPTER 7
CONCLUSIONS

7.1 Summary and Conclusions

This thesis has described the design and implementation of the Associative Rule Memory and has demonstrated its ability to function within a robotic planning system. The motivation for this work is:

1. The need for an evaluation function that ranks alternative robotic actions for a planning step, in a world where many robotic actions can lead to the same effect.

2. The requirement that the evaluation function must efficiently find the optimal robotic actions with respect to the evaluation function for a given desired effect.

3. A desire to model the uncertainty inherent to robotic systems, and incorporate this model into the planning of robotic actions.

Based on these needs, the ARM was designed to embody the following features:

1. The ability to interface with a variety of planning systems through the use of general and specific rules.

2. The storage of tested robotic action/desired effect pairings with probability of effect values.

3. The storage of known symbol relationships, as provided by a user.

4. The ability to extract relationships between symbols that affect POE values and use these relationships to provide predictive POE values for untested robotic action/desired effect pairings.

5. The ability to provide as output, a set of high POE robotic actions that achieve a desired effect, given the desired effect as input.

6. The ability to produce a confidence factor that indicates the training received by the weights of an untested robotic action/desired effect pairing.

The main contributions of this thesis are:

1. The design of a neural network model, called the ARM, that is able to represent a symbolic grammar comprised of a robotic action and effect.

2. The ability of this model to maintain instantiations of the grammar with a real valued number representing the probability that the robotic action achieves the desired effect.

3. A training procedure that guarantees that the ARM will develop accurate POE representations for all specific rules in the training set.

4. A training procedure that develops weighted connections that represent the extent to which agents and actions of a robotic action can work together to achieve a desired effect.

5. A technique for adding higher order nodes when necessary, and pruning them when they are unnecessary.

6. A demonstration that the training procedure builds connections that can be used for predicting POE values for untested specific rules.

7. The addition of known agent relationships to the ARM model through the use of knowledge rules and a confidence factor that provides the user with a measure of confidence in untested specific rules.

8. The demonstration that a Genetic Algorithm can be used to find the minimum energy state of a Boltzmann Machine.

9. The development of the immigration operator for Genetic Algorithms and the demonstration that immigration improves the performance of a GA on functions that possess difficult local optima.

10. The proof that a GA combined with the immigration operator will converge in probability to the global optimum of a cost function.

11. The demonstration that the ARM can function as the Organization level of the Intelligent Machine.

12. The development of a case study based on the Flight Telerobotic Servicer Task Analysis Methodology that demonstrates

    (a) the effectiveness of the training procedure,

    (b) the efficiency of associative recall,

    (c) and the use of the ARM on a complex, target world model.

7.2 Recommendations for Future Research

The following research items can extend the results presented in this thesis.

1. Extending the ARM to multiple planning steps. The ARM is designed to find the optimal robotic actions for a given desired effect. When developing a sequence of planning steps, the set of robotic actions provided by the ARM may require the use of many actors and objects in the world. To reduce the number of resources required by a plan, it may be helpful to have the ARM search for several sequential planning steps at once, under the constraint that each planning step uses the same actor or set of objects.

   One method for accomplishing this is to search several identical ARM models at once, each model provided with a different desired effect as input. Each model, therefore, corresponds to a different planning step. The robotic actions

tested on each model are constrained to use the same resource, and a new cost function is established to represent the sum of the Energy values for each model. By minimizing the Energy function over the set of constrained robotic actions, a set of steps are developed that have high POE values and reduce the number of resources used.

2. Developing a system for providing specific rules. A system is needed to compute POE values for tested robotic actions, and provide these robotic actions to the ARM in the form of specific rules. Testing robotic actions in the environment can be done by a user, or accomplished automatically. For example, Miller [110], began development of an automatic, bottom-up system that interacts with the environment and develops general and specific rules.

3. The development of a complexity model. The ARM models the uncertainty of specific rules. It is also important to have a measure of the complexity of executing a robotic action. A combined measure of uncertainty and complexity can be used to determine optimal robotic actions given a desired effect.

4. The development of concepts using the ARM model. Currently, the ARM develops inhibition between symbols through the training set. Also, inhibition is created by use of the knowledge rules. Prediction using the ARM is based on this combined inter-symbol inhibition. It would be very useful if the ARM model could abstract the symbols that inhibit each other to determine symbol classes that should not be used together. This would provide valuable feedback to the user.

5. Experimenting with different default probability values. The base POE value for an untested rule in this thesis is 0.80, which indicates that most untested specific rules should work well. This allows the ARM to output high POE values for robotic actions that may be quite unlike any tested rule, and thereby

allows the planning system to explore more robotic actions. When planning, it may be desirable to use a more conservative value, such as 0.50, to force the ARM to produce robotic actions that are more similar to tested specific rules. The effect that changing the base POE value has on recalled robotic actions should be experimented with and analyzed.

6. Accelerating the training technique. As mentioned in Chapter 3, it should be possible to use an accelerated training technique, such as constrained conjugate gradient, to reduce the training time required by the ARM. Since the training is performed off-line, however, there may be little need for an accelerated technique.

7. Experimenting with higher order nodes. In Chapter 6, we determine that it may be difficult to develop higher order nodes. Experiments should be performed that reduce the maximum weight allowed on a first order connection. This eases the development of higher order nodes. Experiments should be performed to test the prediction capabilities of the ARM under these conditions.

8. Evaluating immigration on a generational Genetic Algorithm. The experiments performed in this thesis test immigration on a steady state and a restarted GA. The immigration operator should also be tested on a generation GA using the same test suite of functions.

Literature Cited

[1] G. N. Saridis and H. E. Stephanou, "A hierarchical approach to the control of a prosthetic arm," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, no. 6, pp. 407–420, 1977.

[2] "Flight Telerobotic Servicer: Task analysis methodology," tech. rep., Goddard Space Flight Center, Greenbelt, MD, 1989.

[3] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3,4, pp. 189–208, 1971.

[4] A. Newell and G. Ernst, "The search for generality," in *Information Processing 65: Proceedings of IFIP Congress 1965* (W. A. Kalenich, ed.), pp. 17–24, Washington, D. C.: Spartan Books, 1965.

[5] E. D. Sarcerdoti, "Planning in a hierarchy of abstraction spaces," in *Third International Joint Conference on Artificial Intelligence*, (Stanford, CA), pp. 412–422, 1973.

[6] E. D. Sarcerdoti, "The nonlinear nature of plans," in *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, (Tbilisi, Georgia, USSR), pp. 206–214, 1975.

[7] M. Stefik, "Planning with constraints (MOLGEN: part 1)," *Artificial Intelligence*, vol. 16, no. 2, pp. 111–140, 1981.

[8] M. Stefik, "Planning with constraints (MOLGEN: part 2)," *Artificial Intelligence*, vol. 16, no. 2, pp. 141–170, 1981.

[9] A. Tate, "Generating project networks," in *International Joint Conference on Artificial Inteligence*, (Cambridge, MA), 1977.

[10] D. E. Wilkens, "Domain-independent planning: Representation and plan generation," *Artificial Intelligence*, vol. 22, no. 3, pp. 269–302, 1984.

[11] M. J. Rokey, "Remote Mission Specialist: A study in real-time, adaptive planning," *IEEE Transactions on Robotics and Automation*, vol. 6, pp. 455–461, 1990.

[12] G. N. Saridis, "Toward the realization of intelligent controls," *IEEE Proceedings*, vol. 67, no. 8, pp. 1115–1133, 1979.

[13] G. N. Saridis, "Intelligent robotic control," *IEEE Transactions on Automatic Control*, vol. 28, no. 5, pp. 547–556, 1983.

[14] G. N. Saridis, "Control performance as an entropy," *Control Theory and Advanced Technology*, vol. 1, no. 2, 1985.

[15] G. N. Saridis, "On the revised theory of intelligent machines," in *Proceedings of an International Workshop on Intelligent Robots and Systems*, (Tsukuba, Japan), September 1989.

[16] M. C. Moed and G. N. Saridis, "A Boltzmann machine for the organization of intelligent machines," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, pp. 1094–1102, September 1990.

[17] K. P. Valavanis, *A Mathematical Formulation for the Analytical Design of Intelligent Machines.* PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, 1986.

[18] G. N. Saridis and K. P. Valavanis, "Analytical design of intelligent machines," *Automatica*, vol. 24, no. 2, pp. 123–133, 1988.

[19] L. Brownston, R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5.* Addison-Wesley. 1986.

[20] C. Torras I Genis, "Relaxation and neural learning: Points of convergence and divergence," *Journal of Parallel Distributed Computing*, vol. 6, pp. 217–244, 1989.

[21] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural Networks*, vol. 1, no. 1, pp. 17–62, 1988.

[22] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4–22, April 1987.

[23] G. E. Hinton, "Connectionist learning procedures," *Artificial Intelligence*, vol. 40, pp. 185–234, 1989.

[24] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing, Vol. I.* Cambridge, MA: The MIT Press, 1986.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing Volume I* (D. E. Rumelhart and J. L. McClelland, eds.), pp. 318–362, Cambridge, MA: The MIT Press, 1986.

[26] P. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, Cambridge, MA, 1974.

[27] R. J. Williams, "Towards a theory of reinforcement-learning connectionist systems," Tech. Rep. NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA, 1988.

[28] M. W. Roth, "A survey of neural network technology for automatic target recognition," *IEEE Transactions on Neural Networks*, vol. 1, pp. 28–43, 1990.

[29] W. Y. Huang and R. P. Lippmann, "Comparisons between neural net and conventional classifiers," in *Proceedings of the IEEE First International Conference on Neural Networks*, (San Diego, CA), pp. 485–492, 1987.

[30] D. W. Ruck, "Multisensor target detection and classification," Master's thesis, AFIT/GE/ENG, Wright-Patterson AFB, Ohio, 1987.

[31] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, pp. 4–27, 1990.

[32] P. J. Antsaklis, "Neural networks for control systems," *IEEE Transactions on Neural Networks*, vol. 1, p. 148, 1990.

[33] S. Y. Kung and J. N. Hwang, "Neural network architectures for robotic applications," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 5, pp. 641–657, 1989.

[34] G. Josin, "Neural-space generalization of a topological transformation," *Biological Cybernetics*, vol. 59, pp. 283–290, 1988.

[35] H. Miyamoto M. Kawato, T. Setoyama, and R. Suzuki, "Feedback-error-learning neural network for trajectory control of a robotic manipulator," *Neural Networks*, vol. 1, pp. 251–265, 1988.

[36] M. Kawato, Y. Uno, M. Isobe, and R. Suzuki, "Hierarchical neural network model for voluntary movement with application to robotics," *IEEE Control Systems Magazine*, pp. 8–15, April 1988.

[37] K. Goldberg and B. Pearlmutter, "Using a neural network to learn the dynamics of the CMU Direct-Drive Arm II," Tech. Rep. CMU-CS-88-160, Carnegie Mellon University, Pittsburgh, PA, 1988.

[38] D. S. Day, "Towards integrating automatic and controlled problem solving," in *IEEE First International Conference on Neural Networks*, vol. 2, (San Diego, CA), pp. 661–669, 1987.

[39] S. Grossberg, "Adaptive pattern classification and universal recoding, i: Parallel development and coding of neural feature detectors," *Biological Cybernetics*, vol. 23, pp. 121–134, 1976.

[40] S. Grossberg, *Studies of mind and brain: Neural Principles of learning, perception, development, cognition, and motor control.* Boston: Reidel Press, 1982.

[41] S. Grossberg and N. A. Schmajuk, "Neural dynamics of adaptive timing and temporal discrimination during associative learning," *Neural Networks*, vol. 2, no. 2, pp. 79–102, 1989.

[42] D. S. Levine and P. S. Prueitt, "Modeling some effects of frontal lobe damage - novelty and perseveration," *Neural Networks*, vol. 2, no. 2, pp. 102–116, 1989.

[43] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54–115, 1987.

[44] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, pp. 2554–2558, 1982.

[45] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences*, vol. 81, pp. 3088–3092, 1984.

[46] M. Zak, "Terminal attractors for addressable memory in neural networks," *Physics Letters A*, vol. 133, pp. 18–22, 1988.

[47] M. W. Hirsch, "Convergence in neural nets," in *IEEE First International Conference on Neural Networks*, vol. 2, (San Diego, CA), pp. 115–124, 1987.

[48] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, "The capacity of the Hopfield associative memory," *IEEE Transactions on Information Theory*, vol. IT-33, pp. 461–482, July 1987.

[49] S. Amari and K. Maginu, "Statistical neurodynamics of associative memory," *Neural Networks*, vol. 1, no. 1, pp. 63–73, 1988.

[50] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52, pp. 141–152, 1985.

[51] D. S. Touretzky, "Representing conceptual structures in a neural network," in *IEEE First International Conference on Neural Networks*, vol. 2, (San Diego, CA), pp. 279–286, 1987.

[52] G. E. Hinton, "Learning distributed representations of concepts," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 1–12, 1986.

[53] C. P. Dolan and M. G. Dyer, "Symbolic schemata, role binding and the evolution of structure in connectionist memories," in *IEEE First International Conference on Neural Networks*, vol. 2, (San Diego, CA), pp. 287–298, 1987.

[54] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cognitive Science*, vol. 9, pp. 147–169, 1985.

[55] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in Boltzmann machines," in *Parallel Distributed Processing Volume I* (D. E. Rumelhart and J. L. McClelland, eds.), pp. 282–317, Cambridge, MA: The MIT Press, 1986.

[56] H. J. Sussmann, "Learning algorithms for Boltzmann machines," in *Proceedings of the 27th Conference on Decision and Control*, (Austin, TX), pp. 786–791, 1988.

[57] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 4598, pp. 671–680, 1983.

[58] G. E. Hinton and T. J. Sejnowski, "Separating figure from ground with a Boltzmann machine," in *Vision, Brain, and Cooperative Computation* (M. A. Arbib and A. R. Hanson, eds.), pp. 703–724, Cambridge, MA: The MIT Press, 1987.

[59] J. H. M. Korst and E. H. L. Aarts, "Combinatorial optimization on a Boltzmann machine," *Journal of Parallel and Distributed Computing*, vol. 6, pp. 331–357, 1989.

[60] D. E. Rumelhart, P. Smolensky, J. L. McClelland, and G. E. Hinton, "Schemata and sequential thought processes in PDP models," in *Parallel Distributed Processing Volume II* (D. E. Rumelhart and J. L. McClelland, eds.), pp. 7–57, Cambridge, MA: The MIT Press, 1986.

[61] H. Geffner and J. Pearl, "On the probabilistic semantics of connectionist networks," in *IEEE First International Conference on Neural Networks*, vol. 2, (San Diego, CA), pp. 187–195, 1987.

[62] J. Pearl, "Evidential reasoning using stochastic simulation of causal models," *Artificial Intelligence*, vol. 32, pp. 245–257, 1987.

[63] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artificial Intelligence*, vol. 29, pp. 241–288, 1986.

[64] D. S. Touretzky and G. E. Hinton, "Symbols among the neurons: Details of a connectionist inference architecture," in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, (Los Angeles, CA), pp. 239–243, 1985.

[65] D. L. Luenberger, *Linear and Nonlinear Programming, Second Edition.* Addison-Wesley, 1984.

[66] S. Matyas, "Random optimization," *Automatic Remote Control*, vol. 26, pp. 244–251, 1966.

[67] G. N. Saridis, "Expanding subinterval random search for system identification and control," *IEEE Transactions on Automatic Control*, pp. 405–412, 1977.

[68] E. H. L. Aarts and J. H. M. Korst, *Simulated Annealing and Boltzmann Machines.* Chichester: Wiley, 1988.

[69] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions and Bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721–741, November 1984.

[70] J. H. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: The University of Michigan Press, 1975.

[71] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[72] K. A. De Jong, *An Analysis of the Behavior of a class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, MI, 1975.

[73] L. Davis, "Job shop scheduling with genetic algorithms," in *Proceedings of an International Conference on Genetic Algorithms*, pp. 136–140, 1985.

[74] L. Davis, "Applying adaptive algorithms to epistatic domains," in *Proceedings of the 9th International Joint Conference On Artificial Intelligence*, pp. 162–164, 1985.

[75] D. E. Glover, "Solving a complex keyboard configuration problem through generalized adaptive search," in *Genetic Algorithms and Simulated Annealing* (L. Davis, ed.), London: Pitman, 1987.

[76] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, (Cambridge, MA), pp. 14–21, 1987.

[77] J. J. Grefenstette and J. E. Baker, "How genetic algorithms work: A critical look at implicit parallelism," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 20–27, 1989.

[78] D. E. Goldberg, "Sizing populations for serial and parallel genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 70–79, 1989.

[79] L. J. Eshelman, R. A. Caruana, and J. D. Schaffer, "Biases in the crossover landscape," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 10–19, 1989.

[80] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. Richards, "Punctuated equilibria: A parallel genetic algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, (Cambridge, MA), pp. 148–154, 1987.

[81] D. E. Brown, C. L. Huntley, and A. R. Spillane, "A parallel genetic heuristic for the quadratic assignment problem," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 406–415, 1989.

[82] H. Mülenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 416–421, 1989.

[83] M. Gorges-Schleuter, "ASPARAGOS an asynchronous parallel genetic optimization strategy," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 422–427, 1989.

[84] C. C. Pettey and M. R. Leuze, "A theoretical investigation of a parallel genetic algorithm," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 398–405, 1989.

[85] R. Tanese, "Distributed genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 434–439, 1989.

[86] H. J. Antonisse and K. S. Keller, "Genetic operators for high level knowledge representation," in *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, (Cambridge, MA), pp. 69–76, 1987.

[87] J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. V. Gucht, "Genetic algorithms for the traveling salesman problem," in *Proceedings of an International Conference on Genetic Algorithms*, pp. 160–168, 1985.

[88] L. Davis and S. Coomb, "Genetic algorithms and communication link speed design: Theoretical considerations," in *Proceedings of an International Conference on Genetic Algorithms and their Applications*, (Cambridge, MA), pp. 252–256, 1987.

[89] J. D. Schaffer, R. A. Caruana, L. J. Eschelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 51–60, 1989.

[90] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, 1987.

[91] L. Davis, "Adapting operator probabilities in genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 61–69, 1989.

[92] R. A. Caruana and J. D. Schaffer, "Representation and hidden bias: Gray vs. binary coding for genetic algorithms," in *Proceedings of the Fifth Int. Conf. on Machine Learning*, pp. 153–161, 1988.

[93] R. B. Hollstien, *Artificial genetic adaptation in computer control systems.* PhD thesis, University of Michigan, Ann Arbor, MI, 1971.

[94] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, (Cambridge, MA), pp. 41–49, 1987.

[95] K. Deb and D. E. Goldberg, "An investigation of niche and species formation in genetic function optimization," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 42–50, 1989.

[96] G. N. Saridis, "Analytic formulation of the principle of increasing precision with decreasing intelligence for intelligent machines," *Automatica*, vol. 25, no. 3, pp. 461–467, 1989.

[97] A. Levis, "Human organizations as distributed intelligence systems," in *Proceedings of the First IFAC-IMACS Symposium on Distributed Intelligence Systems*, (Varna, Bulgaria), 1988.

[98] G. N. Saridis, "Entropy formulation for optimal and adaptive control," *IEEE Transactions on Automatic Control.* vol. 33, no. 8, pp. 713–721, 1988.

[99] G. N. Saridis, *Self-Organizing Controls of Stochastic Systems.* New York: Marcel Dekker, 1977.

[100] J. R. Birk and R. B. Kelley, "An overview of the basic research needed to advance the state of knowledge in robotics," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 8. pp. 575–579, 1981.

[101] F. Y. Wang and G. N. Saridis, "A model for coordination of intelligent machines using Petri nets," in *IEEE Symposium on Intelligent Control*, (Washington, D.C.), pp. 28–33, August 1988.

[102] F. Y. Wang, *A Coordination Theory for Intelligent Machines.* PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, 1990.

[103] K. J. Kyriakopoulos and G. N. Saridis, "Collision avoidance of mobile robots in a non-stationary environment," *Control Systems Magazine*, June 1991.

[104] E. T. Jaynes, "Information theory and statistical mechanics," *Physical Review*, vol. 106, no. 4, pp. 620–630, 1957.

[105] *American Heritage Dictionary of the English Language.* 1969.

[106] R. C. Conant, "Laws of information which govern systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, no. 4, pp. 240–255, 1976.

[107] T. Vamos, "Metalanguages - conceptual model: Bridge between machine and human intelligence," in *Proceedings of the 1st International Symposium on AI and Expert Systems*, pp. 237–287. 1987.

[108] A. N. Kolmogorov, "On some asymptotic characteristics of completely bounded metric systems," *Dokl Akad Nank, SSSR*, vol. 108, no. 3, pp. 385–389, 1956.

[109] G. Zames, "On the metric complexity of causal linear systems, ϵ-entropy and ϵ-dimension for continuous time," *IEEE Transactions on Automatic Control*, vol. 124, pp. 222–230, 1979.

[110] S. A. Miller, "PRIME: A bottom-up approach to probabilistic rule development," Tech. Rep. CIRSSE 55, Rensselaer Polytechnic Institute, Troy, NY, 1990.

APPENDIX A
SYMBOL DEFINITIONS

This appendix presents the list of the agent symbols used in the thesis and a description of what each symbol represents. These descriptions are modified slightly from the NASA Flight Telerobotic Servicer Task Analysis Methodology [2].

- Telerobots
  - Dextrous Manipulators
    - *EVA*. An Extravehicular Astronaut. It is considered to function as light, dextrous manipulator.
    - *FTS*. The Flight Telerobotic Servicer. This is a dextrous manipulator, and can manpulate and lift relatively light objects.
    - *GDMS*. A dextrous manipulator which can manpulate and lift light or heavy objects.
    - *JRMS*. Another dextrous manipulator which can manpulate and lift light objects.
    - *SPDM*. Another dextrous manipulator which can manpulate and lift light objects.
    - *MRMS*. Another dextrous manipulator which can manpulate and lift light or heavy objects.
  - Positioners
    - *SRMS*. The Shuttle Remote Manipulator System. This telerobot is used to position light objects.
    - *SRMS*. The Shuttle Remote Manipulator System. This telerobot is used to position light objects.

    * *SSRMS.* The Space Station Remote Manipulator System. This telerobot is also used to position light objects.
    * *APS.* The Automatic Positioning System. This telerobot is used to position heavy objects.
    * *ATD.* The Automatic Translation Device. This telerobot is used to position heavy objects.
  - Transporters
    * *CETA.* The Crew and Equipment Transportation Aid. This telerobot is used to transport the EVA and moderately heavy equipment.
    * *MMU.* This telerobot is used to transport light manipulators and light to moderately heavy equipment.
    * *MT.* The Mobile Transporter. This telerobot is used to transport light to moderately heavy equipment.
    * *OMV.* The Orbital Maneuvering Vehicle. This system is used to transport light to very heavy equipment.
    * *ORBITER.* This system is used to transport light to very heavy equipment.
- Tools
  - Fixturing
    * *BOLTER.* A tool which bolts an object to another.
    * *CLAMP-H.* A clamp used to fixture heavy or large objects.
    * *CLAMP-L.* A clamp used to fixture small or light objects.
    * *CLAMP-M.* A clamp used to fixture medium size and weight objects.
    * *GLUEGUN.* A tool which glues an object to another using strong, yet removeable, adhesive.

  * *GRAPPLER-H.* A grappler used to fixture heavy or large objects.
  * *GRAPPLER-L.* A grappler used to fixture small or light objects.
  * *GRAPPLER-M.* A grappler used to fixture medium size and weight objects.
  * *PINS-H.* A pinning device used to fixture heavy or large objects.
  * *PINS-L.* A pinning device used to fixture small or light objects.
  * *PINS-M.* A pinning device used to fixture medium size and weight objects.
  * *WELDER.* A tool which attaches an object to another using strong adhesive.

- Dexfixturing
  * *PRYBAR.* A tool which aids in the separation of one object from another.
  * *SEPARATOR.* A tool which aids in the separation of one object from another.
  * *DEMATOR.* A tool which aids in the separation of one object from another.

- Actuating
  * *TOOLSET0.* A toolset used by light manipulators.
  * *TOOLSET1.* A toolset used by light manipulators.
  * *TOOLSET2.* A toolset used by light manipulators.
  * *TOOLSET3.* A toolset used by heavy manipulators.
  * *TOOLSET4.* A toolset used by heavy manipulators.
  * *TOOLSET5.* A toolset used by two cooperating manipulators.
  * *TOOLSET6.* A toolset used by two cooperating manipulators.

- Carriers and Parts
  - *CARRIER-H.* A carrier that objects are fixtured to before transporting.
  - *CARRIER-L.* A carrier that only small or light objects are fixtured to before transporting.
  - *CARRIER-M.* A carrier that small, light or medium size or weight objects are fixtured to before transporting.
  - *PALLET-H.* A carrier that objects are fixtured to before transporting.
  - *PALLET-L.* A carrier that only small or light objects are fixtured to before transporting.
  - *PALLET-M.* A carrier that small, light or medium size or weight objects are fixtured to before transporting.
  - *ORU.* An Orbital Replacement Unit. This is a module used in the Space Station Environment. Its function can vary.
  - *TRUSS.* This is the base Space Station structure.
- Sites
  - *AIRLOCK.* The Shuttle airlock.
  - *AWP.* The Assembly Work Platform. This platform is located in the Space Station enviroment and serves as a place for telerobots to assemble components of the Space Station.
  - *CARGO-BAY.* The Cargo Bay of the Space Shuttle.
  - *MSC.* The Mobile Servicing Center. This platform is located in the Space Station environment and serves as a place for telerobots to service and repair components of the Space Station.

APPENDIX B
SYMBOL CLASS HIERARCHY

This appendix presents a diagram of the hierarchy of symbol classes used in the thesis. The diagram also presents the agent symbols that belong to each symbol class.

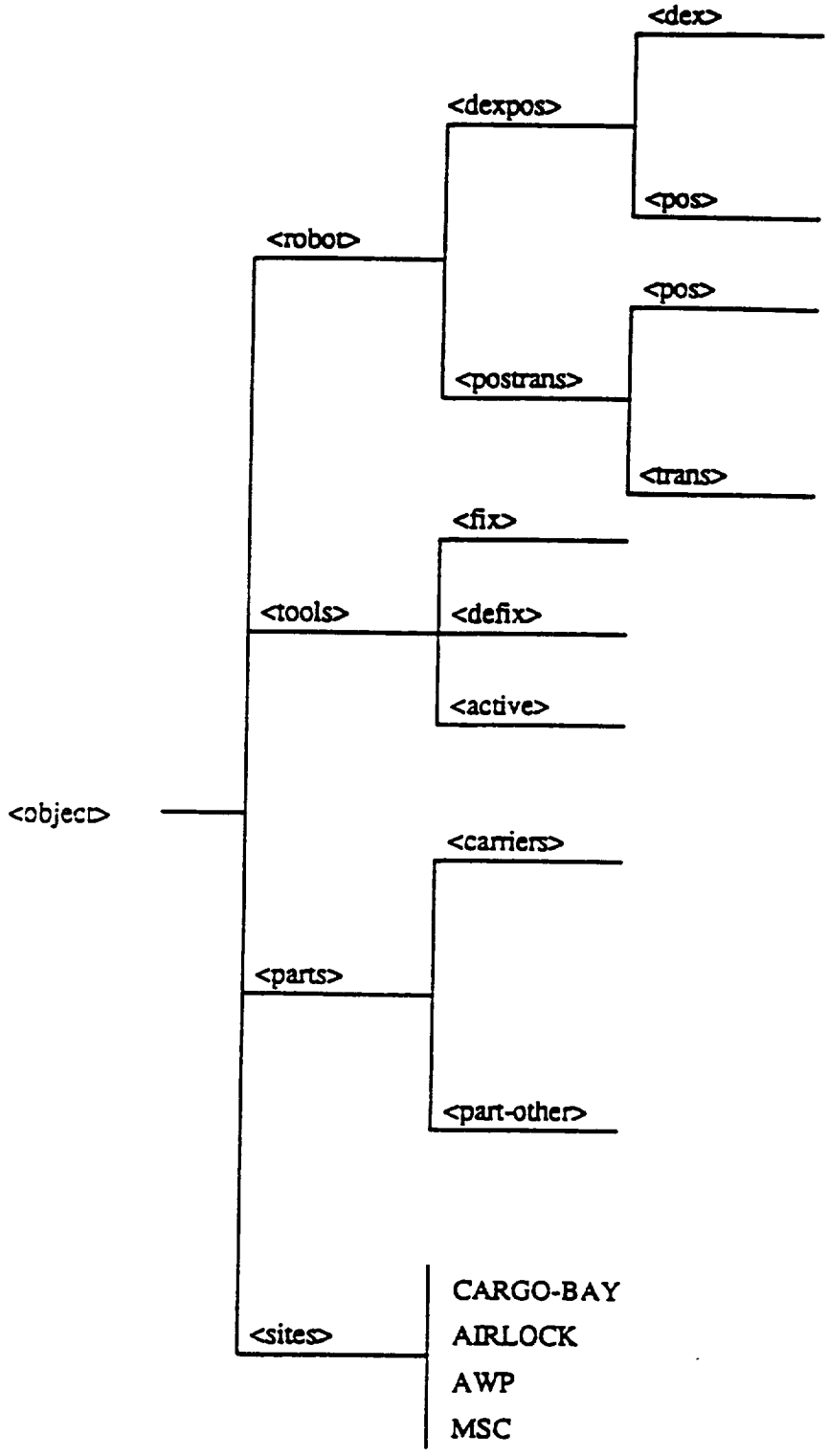


Figure B.1: Classification of agents in the world model

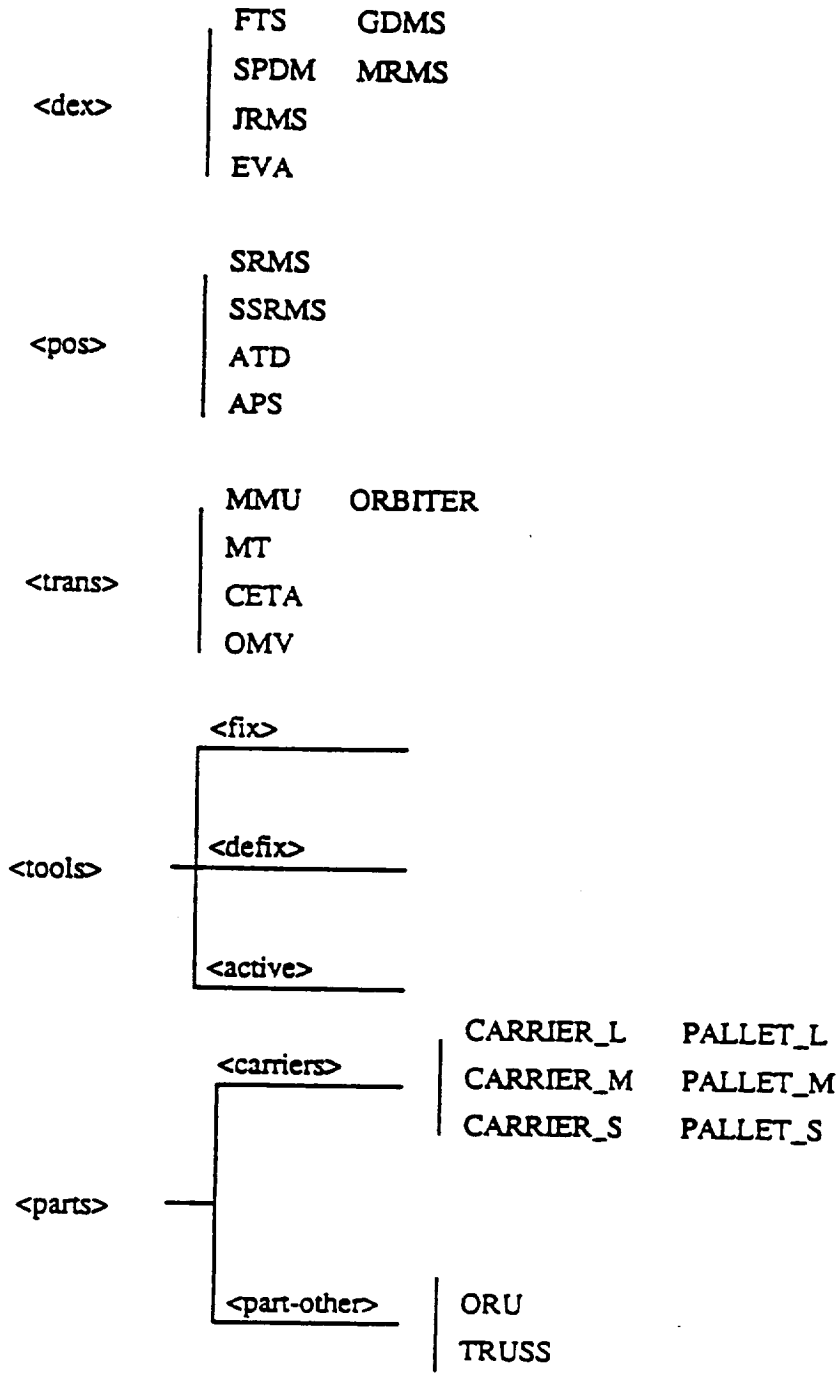


Figure B.2: Classification of agents in the world model, cont'd

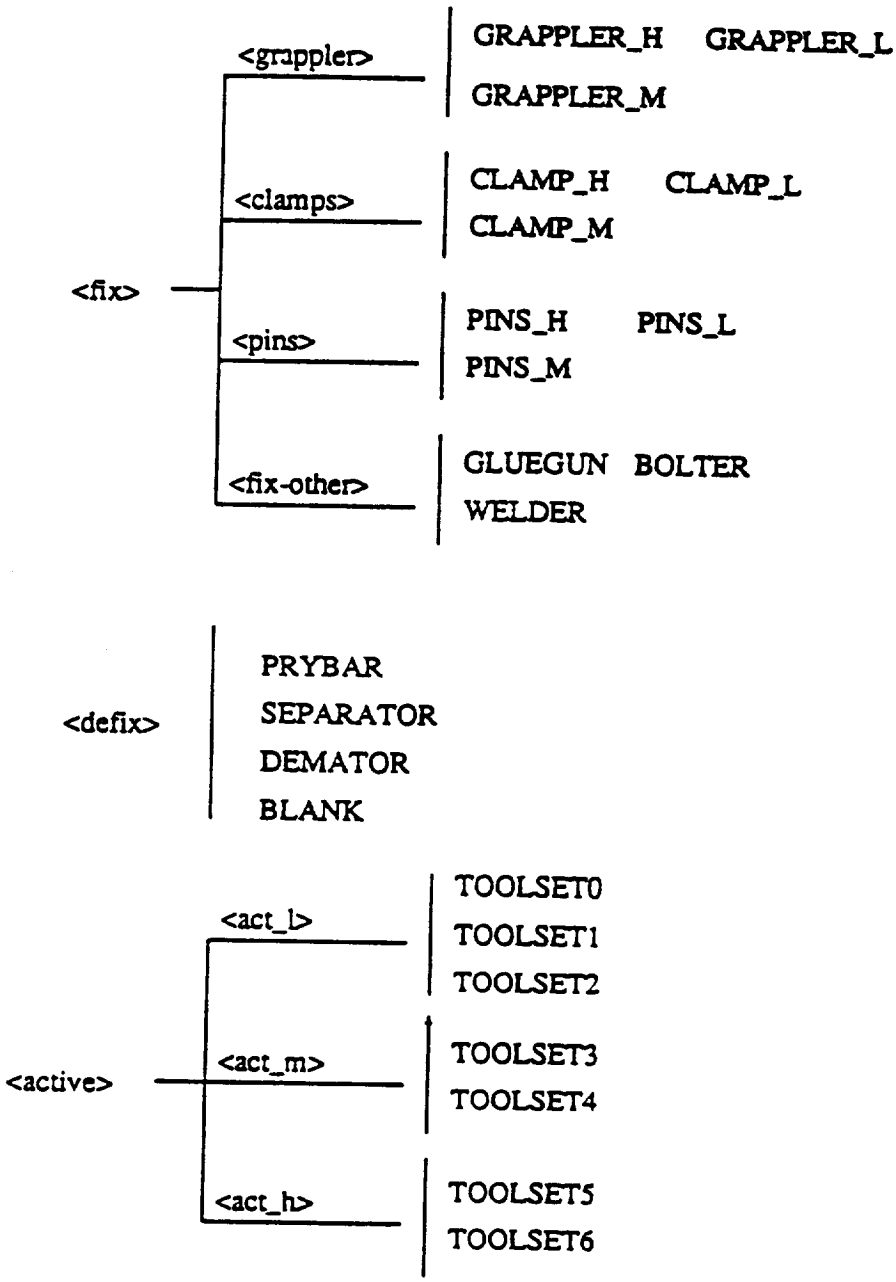


Figure B.3: Classification of agents in the world model, cont'd